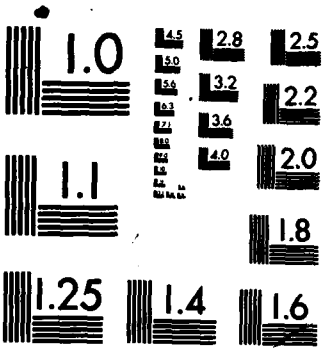
AD-A161 854 A STUDY OF RELAXATION TECHNIQUES FOR THE TRANSIENT 1/2
ANALYSIS OF DIGITAL CI (U) ILLINOIS UNIV CHAMPAIGN
COGNITIVE PSYCHOPHYSIOLOGY LAB W CHIA 03 JUN 85
UNCLASSIFIED UILU-ENG-85-2203 N00014-84-C-0149 F/G 9/3 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

COORDINATED SCIENCE LABORATORY

A STUDY OF RELAXATION TECHNIQUES FOR THE TRANSIENT ANALYSIS OF DIGITAL CIRCUITS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): R-report # 1028	5. MONITORING ORGANIZATION REPORT NUMBER(S): N/A
6a. NAME OF PERFORMING ORGANIZATION: Coordinated Science Laboratory, Univ. of Ill. / 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research
6c. ADDRESS (City, State and ZIP Code): 1101 W. Springfield Avenue, Champaign, Illinois 61801	7b. ADDRESS (City, State and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research / 8b. OFFICE SYMBOL (If applicable): N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Contract # N00014-84-C-0149
8c. ADDRESS (City, State and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): A Study of Relaxation Techniques for the Transient Analysis of Digital Circuits	
12. PERSONAL AUTHOR(S): Wei-Kong Chia	
13a. TYPE OF REPORT: Technical / 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): June 3, 1985 / 15. PAGE COUNT: 147
16. SUPPLEMENTARY NOTATION: N/A	
17. COSATI CODES — FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Microelectronics, standard, tearing, relaxation, decomposition, exploitation.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In the VLSI microelectronics era, the cost of the immense CPU time and memory storage for a "standard" circuit simulator has become prohibitive. In order to achieve dramatic improvement in the performance of the circuit simulator, there are two principal points of departure from the "standard" simulation approach, namely, "tearing" decomposition and "relaxation" decomposition.

This research is to study the numerical convergence and stability properties of several of the relaxation algorithms that have been proposed for the simulation of VLSI circuits. The time-point Gauss-Seidel method with prediction, the exploitation of latency and event scheduling algorithms are implemented into a general purpose circuit simulator SLATE-R (a Simulator with Latency and Tearing -- Relaxed version). The performance of the SLATE-R program in the analysis of various types of integrated circuit technologies is studied.

(Keywords included)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) / 22c. OFFICE SYMBOL: NONE

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

A STUDY OF RELAXATION TECHNIQUES FOR THE TRANSIENT ANALYSIS OF DIGITAL CIRCUITS

BY

WEI-KONG CHIA

B.S., National Cheng Kung University, 1976
M.S., National Cheng Kung University, 1978

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1984

Urbana, Illinois

A STUDY OF RELAXATION TECHNIQUES FOR THE TRANSIENT ANALYSIS OF DIGITAL CIRCUITS

Wei-Kong Chia, Ph.D

Department of Electrical Engineering
University of Illinois at Urbana-Champaign, 1984

In the VLSI microelectronics era, the cost of the immense CPU time and memory storage for a 'standard' circuit simulator has become prohibitive. In order to achieve dramatic improvement in the performance of the circuit simulator, there are two principal points of departure from the 'standard' simulation approach, namely, 'tearing' decomposition and 'relaxation' decomposition.[1]

This research is to study the numerical convergence and stability properties of several of the relaxation algorithms that have been proposed for the simulation of VLSI circuits. The time-point Gauss-Seidel method with prediction, the exploitation of latency and event scheduling algorithms are implemented into a general purpose circuit simulator SLATE-R (a Simulator with Latency and Tearing —Relaxed version). The performance of the SLATE-R program in the analysis of various types of integrated circuit technologies is studied.

---

[1] Compared with the conventional techniques, the tearing decomposition is just some special reordering strategy, therefore, it shares the same numerical properties with the conventional techniques. However, the relaxation decomposition processes one subcircuit at a time and relaxes all other subcircuits, therefore, it is characterized by completely different numerical properties.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor T. N. Trick, my dissertation advisor. With his thorough knowledge of the field, he provided invaluable guidance and encouragement through the course of this research. Also his great personality provided me with support and inspiration in my life throughout my graduate study. I would also like to thank Professor I. N. Hajj for many helpful discussions and suggestions. I also wish to thank Professor J. H. Patel for his helpful suggestions as a member of my dissertation committee.

I am indebted to my wife, Chiou-Sheng, for her support and understanding in all respects throughout my graduate study.

Finally, I would like to thank my parents, especially my mother, who passed away two years ago, for their love and support through the past years.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Circuit simulation has become a significant tool in the design of integrated circuits. 'Standard' circuit simulators, such as SPICE2 [1] and ASTAP [2], substantially include the following four algorithmic techniques :

1) Stiffly stable implicit integration methods, such as the backward Euler or Trapezoidal formulas, which replace the original system of nonlinear differential-algebraic equations, describing the behavior of the circuit, into a system of nonlinear algebraic equations.

2) Automatic control of the time step h by using approximation differentiation to estimate the local truncation error so as to ensure accuracy of the solution.

3) The quadratically convergent Newton-Raphson method to solve the system of the nonlinear algebraic equations by iteratively solving a sequence of linear equations.

4) Sparse Gaussian elimination methods to solve the linear algebraic equations in each Newton-Raphson iteration.

Because of the good numerical stability and accuracy properties of the implicit integration methods, these circuit simulators can handle a wide variety of ordinary differential equations very well. Also with the variable time step size control technique, these methods have a simulation speed advantage in stiff systems (widely separated eigenvalues), because one can vary the integration step size according to the rate of change of the response and not encounter numerical stability problems. For nonlinear equations, Newton-Raphson iteration can be used to achieve convergence over a wide range of integration step sizes. Finally, modified nodal methods are used to formulate the circuit equations because they are efficient and result in sparse arrays. Thus, in order to minimize the number of numerical operations, these equations are solved by means of sparse matrix techniques.

Although these circuit simulators are very efficient, their memory and CPU requirements typically limit their use to a few hundred transistors. However, with the rapid growth in the scale, measured in device count, of integrated circuits being designed in the VLSI microelectronics era, the cost of the immense CPU time and memory storage for "standard" circuit simulators has become prohibitive. In order to achieve even faster circuit simulation with less memory requirements, one must take advantage of some of the special features of these circuits, such as (a) the repetitiveness of the circuit (many gates are of the same type), (b) the different levels of activity of the various gates in a given time interval, and (c)

the almost one-way propagation of the signal in many gates. Hence a series of new generation methods of circuit simulation, such as MOTIS [3], MOTIS-C [4], DIANA [5], SPLICE [6] and MACRO [7] have been developed. These new simulators, in their quest for speed, eliminated one or more of the principal features of the "standard" simulator in order to make a favorable trade-off between cost (i.e., the CPU time and memory storage) and resolution (i.e., the attainable level of detail). The program SLATE [8] takes advantage of items (a) and (b) above by using node tearing to partition the circuit into subcircuits so that the sparse matrix pointers only have to be generated and stored for each different type of gate and not for all gates. Secondly, gates that are latent in a given time interval can be bypassed in the solution of the circuit in that time interval. These improvements have resulted in approximately a factor of two improvement in both memory and CPU requirements.

However, in order to achieve more dramatic improvement in the performance of the circuit simulator, one must take advantage of the one-way propagation of the signal in most gates, that is, the response of a gate has little effect on its input signals, or in other words, there is little feedback to the input nodes. When this feedback coupling is sufficiently weak, one can relax certain variables in order to decouple the gates.

There are two principal points of departure from the "standard" simulation approach which may be taken at any of the three main levels of circuit simulation (i.e., the time level, nonlinear equation

level and the linear equation level), namely, "tearing" decomposition and "temporal" decomposition. As to these two techniques, the former aims to retain the convergence and stability properties of the "standard" method, while the latter is related to the so-called "relaxation" or "indirect" method [9,10], and is characterized by completely different convergence and stability properties.

Ever since the development of the MOTIS program in 1975, the first relaxation-based nonlinear time-domain transient circuit simulator, there have been a series of relaxation-based simulators developed. However, some of these simulators suffer from serious numerical properties. In the era of VLSI circuits, the relaxation-based techniques seem to be an inevitable tendency [11] in order to achieve the speed necessary to analyze large circuits. Thus, a complete, deterministic study of the numerical stability and convergence properties of relaxation methods is a valuable and interesting research topic.

This research is concerned with the investigation of the numerical stability and convergence properties of the two commonly used relaxation methods, waveform and time-point (Gauss-Seidel), as a function of the degree of coupling. The modified waveform relaxation method for the simulation of VLSI circuits in the time domain is studied. This approach is similar to the waveform relaxation method in RELAX [9]. However, the entire time interval is separated into small time windows. Instead of a sweeping iteration in the entire time interval [0, T], the sweep iteration is processed sequentially in

each window and the waveforms are concatenated. The experimental results [12] show that the modified technique can get better resolution with reduced cost, thus permitting larger systems to be simulated.

In the transient analysis of MOS circuits in which the floating gate to drain capacitance is modeled, the pole-splitting phenomenon [13] occurs when the transistors are active. The stiffness of the system is determined by the degree to which the poles split. In this research, the numerical properties of the waveform relaxation method are studied for the analysis of linear stiff systems. We examine the numerical properties by means of the test circuit which was generated by linearizing the model of a cascade of two inverters in which the transistors are assumed to be active. When the window size is shrunk to the time step size, the waveform relaxation method is equivalent to a time-point relaxation method. The time-point relaxation method considered in this research is the modified Gauss-Seidel method [9], which uses a forward predictor for the unsolved variables. It has been demonstrated to be an efficient technique in the timing analysis of MOS circuits. The numerical properties of the modified Gauss-Seidel method are also observed for linear stiff systems.

The SLATE program is modified to implement the relaxation techniques in this research. Chapter 2 describes briefly the different relaxation techniques used in electrical simulation. In Chapter 3 the modified Gauss-Seidel technique and its numerical properties for the analysis of a linear stiff system are discussed. An introduction of

the waveform relaxation method (WRM) and the modified window waveform relaxation method and its numerical properties are discussed in Chapter 4. The experimental results of the program SLATE-R (a Simulator with Latency and Tearing - Relaxed version) are shown in Chapter 5. Chapter 6 describes the implementations of the SLATE-R program. Finally, the conclusions are in Chapter 7.

CHAPTER 2

REVIEW OF SIMULATION TECHNIQUES

Standard circuit simulators have proven to be reliable and effective when the size of the circuit is limited to several hundred transistors. As the size of the circuit increases, the primary memory storage and CPU time used by these simulators increase rapidly [14] despite the use of sparse matrix techniques. In order to simulate LSI and VLSI circuits, a number of techniques have been used to improve the performance of the standard circuit simulators. Basically, these nonstandard simulators can be meaningfully classified by the decomposition techniques.

Decomposition refers to the technique that subdivides the whole set of the system equations into several subsets. Decomposition can be taken at any of the three main levels of circuit simulation (i.e., the time level, nonlinear equation level and linear equation level). Actually, the system of equations, no matter at what level it is, is processed by a decomposition technique as a composition of several systems with interactions between them. Once the system is decomposed into subsystems, the technique of solving each subsystem is identical to the conventional numerical approaches.

In this chapter, we will briefly review the analysis techniques used in conventional circuit simulation. Then we will describe the basic concepts and properties of two systematic approaches to

accomplish system decomposition.

2.1 Standard Circuit Simulators

The nonlinear algebraic-differential equations which describe the performance of the integrated circuits are generally of the following form :

$$f(x(t),\dot{x}(t),u)=0 \tag{2.1}$$

$$E(x(0)-x_o)=0 \tag{2.2}$$

where $x \in R^p$ is the unknown variable at time t with the given initial value x_o; $\dot{x}$ is the time derivative of x at time t; $u \in R^r$ is the vector of all the inputs and possibly their time derivatives; $f : R^p \times R^p \times R^r \rightarrow R^p$ is a continuous function; and $E \in R^{nxp}$, $n \leq p$ is a matrix of rank n such that E(x(t)) is the state of the system at time t. Let $\{ t_i ; i = 0,1,\ldots,N \}$ denote a sequence of increasing time points selected by the simulator with $t_o = 0$ and $t_N = T$, where T is the given simulation time interval.

By using an implicit integration method, the system of equations (2.1) is transformed into a discrete time sequence of nonlinear algebraic equations. At each time point t_i, the corresponding algebraic equation can be written as

$$g(x^i)=0 \tag{2.3}$$

where x^i denotes the computed value of $x(t_i)$.

The solution of (2.3) is obtained by applying the Newton-Raphson method. At each iteration in the Newton-Raphson method, the resulting linearized equations are of the form:

$$A\,x = b \tag{2.4}$$

The Newton-Raphson iteration is carried out until the convergence is achieved or the iteration limit is exceeded. At each iteration count, the function and Jacobian matrix evaluations are necessary to construct the coefficient matrix A in Equation (2.4). Because in the circuit simulation environment, the matrix A is usually very sparse; hence, the Gaussian elimination method is implemented by using a sparse matrix technique to reduce the computational operations.

It is very important to exploit the sparse technique, since the computational complexity of the Gaussian elimination method applied to an nxn full matrix is proportional to n^3 while the computational complexity of the Gaussian elimination method with sparse techniques in on the average [1] proportional to n^{α} ; $\alpha \in [1.2, 1.5]$. Figure 2.1 shows the hierarchical organization of conventional numerical methods for time domain simulation.

As the size of the circuit increases beyond several hundred transistors, sparse techniques alone are not enough to provide simulation results in a reasonable time. In the next section we will

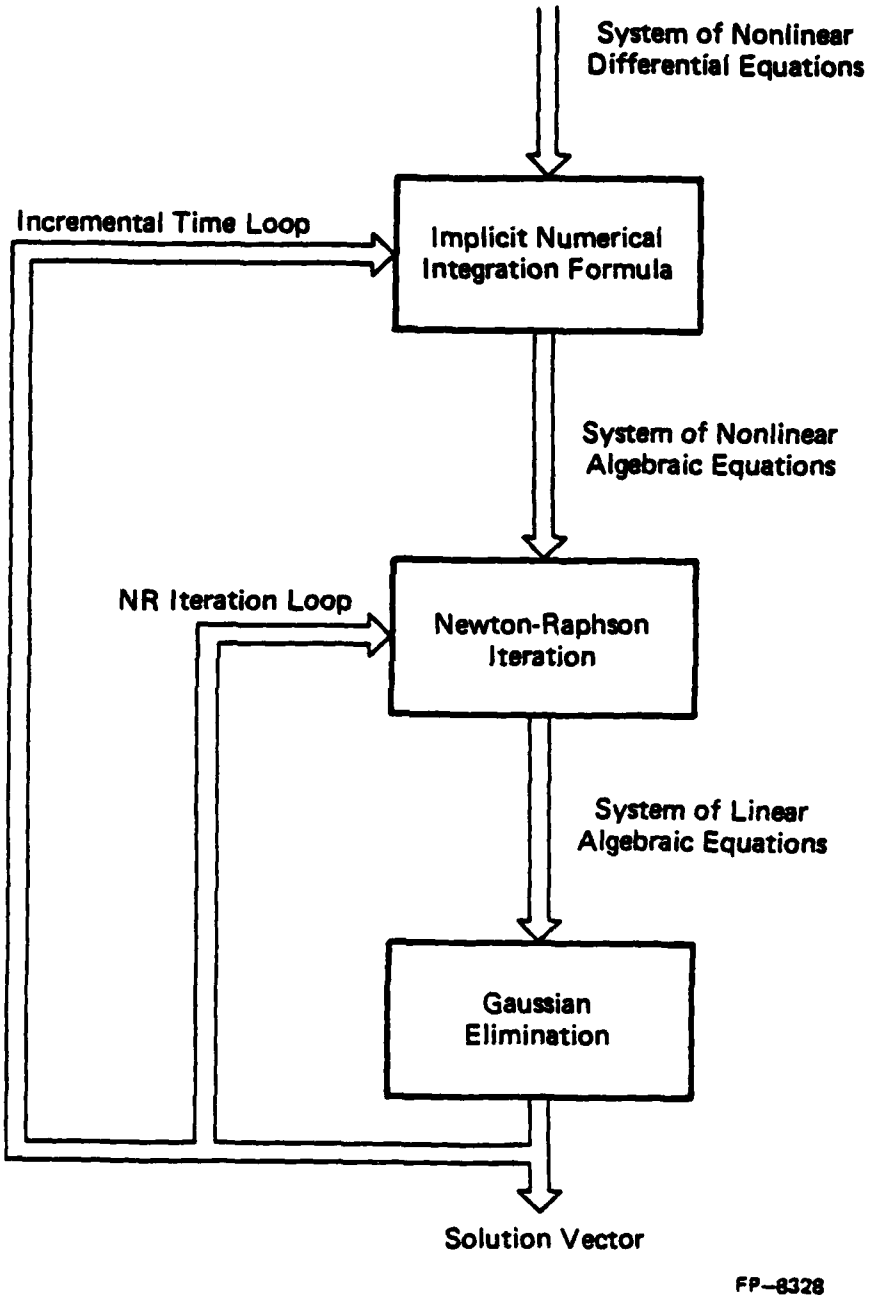


Fig. 2.1 Hierachical Organization of Conventional Circuit Simulation.

describe the features of system decomposition, which have led to the development of several new generations of circuit simulators.

2.2 System Decomposition

The new generation of circuit simulators for large digital circuits uses two principles, namely the 'tearing' decomposition and the 'relaxation' decomposition [15]. Compared with the conventional techniques, the tearing approach is just some special reordering strategy. Therefore, the computational operations of the tearing approach depends mainly on the structure of the system. If the system structure is not sparse or when the block structure of the system can not be exploited, this approach does not offer any benefit over conventional techniques. In other words, the tearing method will be powerful only if it is combined with some other strategies such as the exploitation of latency and the exploitation of the repetitiveness of a limited number of subcircuits.

In contrast, the techniques classified as relaxation methods are characterized by completely different convergence and stability properties. We will discuss the numerical properties of the relaxation techniques in detail in Chapter 3 and Chapter 4.

These two different approaches to decomposition techniques will be described in more detail in the next sections.

2.3 Tearing Decomposition

The idea of tearing decomposition techniques is to select a set of tearing variables to separate the entire system into several subsystems. There are two different approaches to tearing decomposition, namely, branch tearing and node tearing. The former uses tearing branches as the tearing variables (Figure 2.2) while the latter uses tearing nodes as the tearing variables (Figure 2.3). The entire system is torn apart into several subsystems by removing these tearing variables.

Algebraically, the branch tearing method is equivalent to a special reordering of the hybrid analysis equations, whereas the node tearing method is equivalent to a special reordering of nodal analysis equations. However, both methods result in a Bordered Block Diagonal (BBD) matrix structure (Figure 2.4a). Each block corresponds to a subsystem, and the border corresponds to the interconnections. The more general matrix form that is suitable for tearing decomposition is the Border Block Lower Triangular (BBLT) structure (Figure 2.4b).

Tearing decomposition of linear algebraic equations can be implemented through the Block LU Factorization [16]. Let us consider the system of equations shown in Figure 2.5, i.e.,

$$A\,x = b$$

where $x = [\, v \; w \,]^T \in R^n$ is the vector of the unknown variables and w is the vector of the tearing variables. The solution strategy has

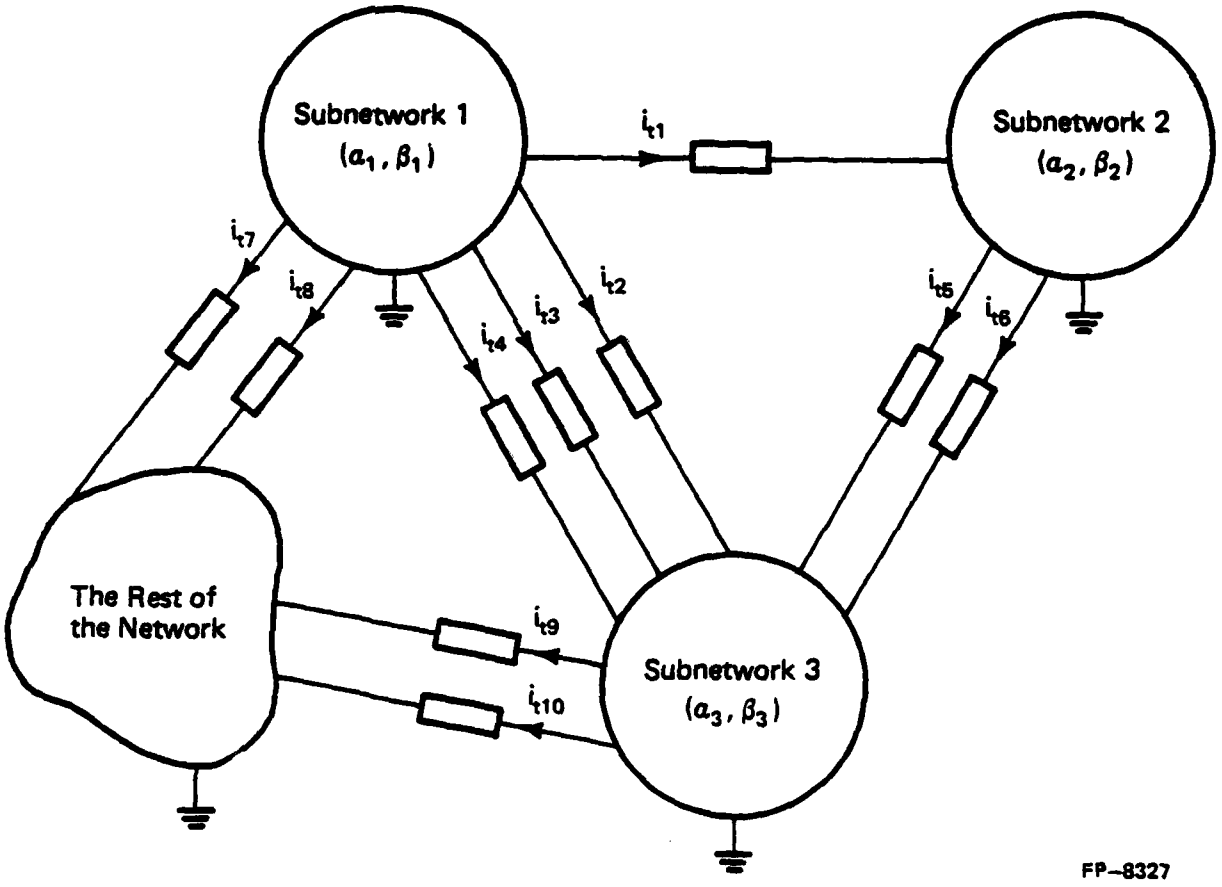


Fig. 2.2 Example of a Network Partitioned into Three Subnetworks by the Branch Tearing Method.

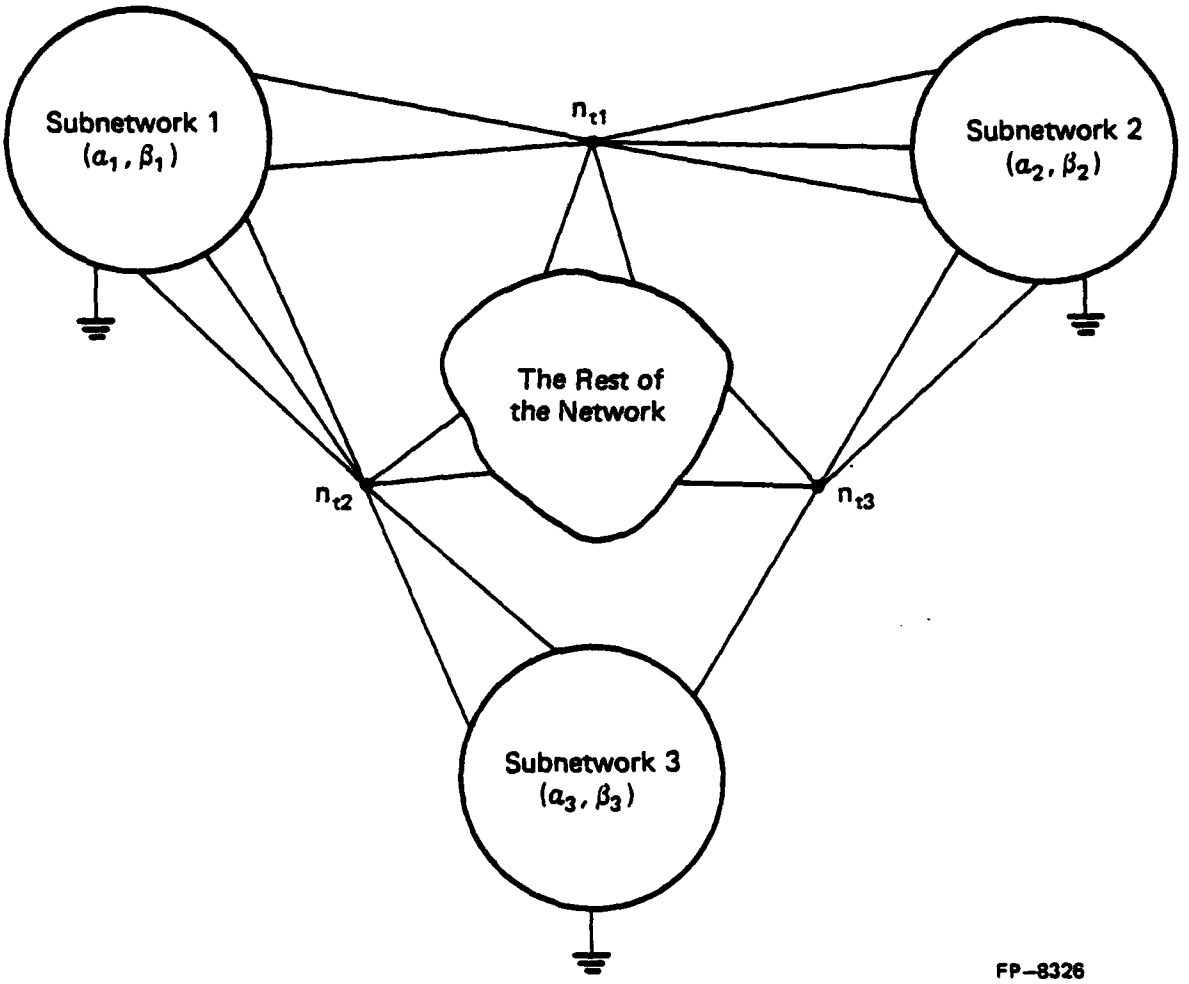


Fig. 2.3 Example of a Network Partitioned into Three Subnetworks by the Node Tearing Method.

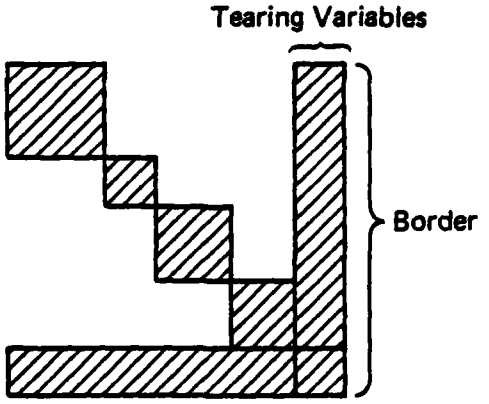


Fig. 2.4a Bordered Block Diagonal (BBD) Form of a Matrix.

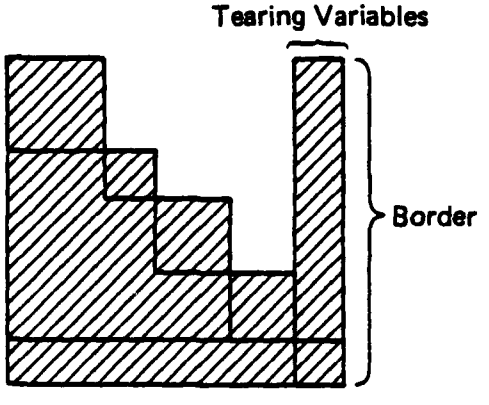


Fig. 2.4b Bordered Block Lower Triangular (BBLT) Form of a Matrix.

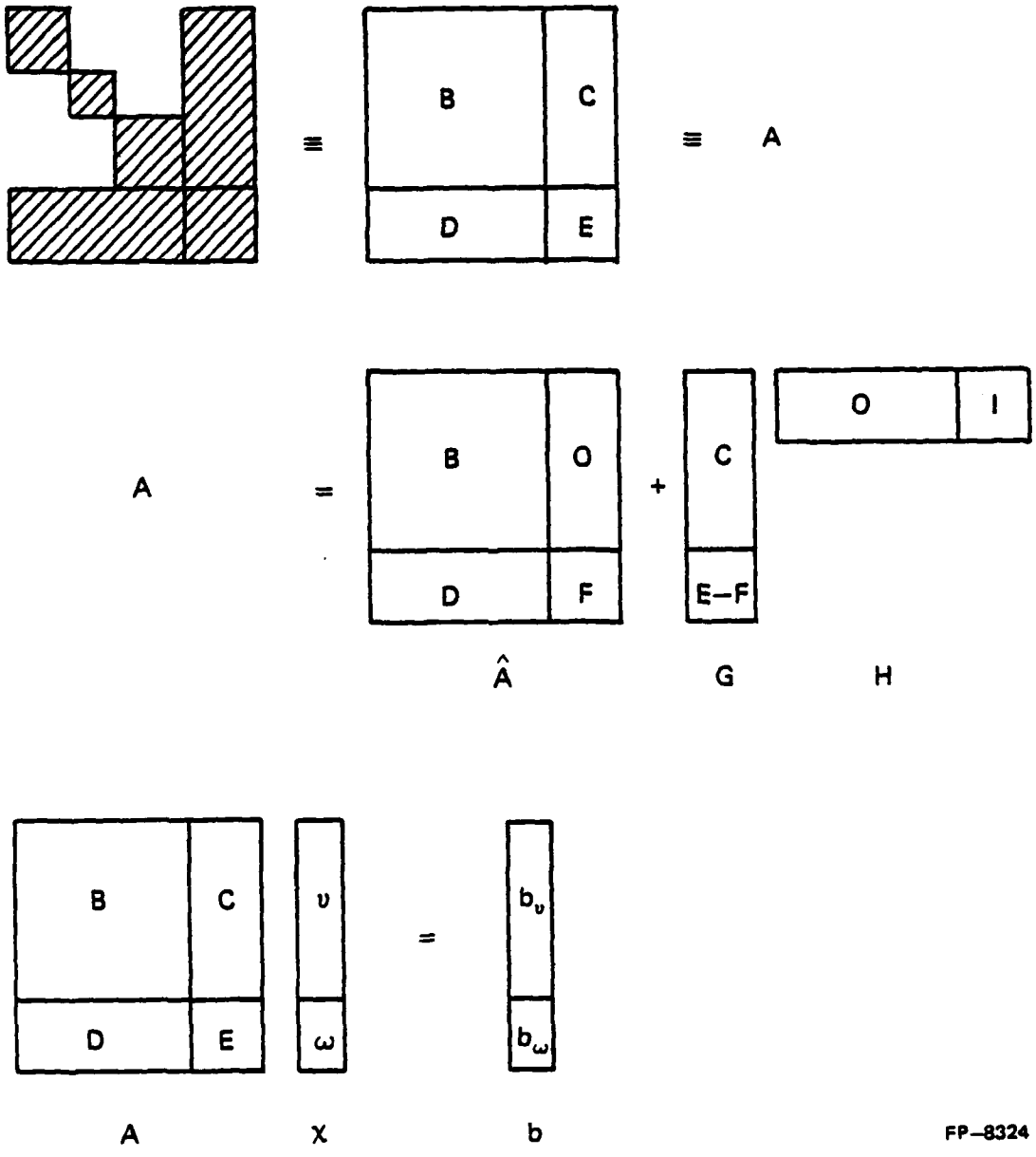


Fig. 2.5 Various Terms Associated with Block LU Factorization and Tearing Algorithm.

been to first eliminate the variables v from the system of equations to obtain the following reduced subsystem from which the value of the tearing variable w is achieved.

$$(E - DB^{-1}C)w = b_w - DB^{-1}b_v$$

where the corresponding meanings of the matrices and vectors are given in Figure 2.5. The computed solution of w is then used to compute the solution of v blockwise. Some other techniques, such as the Tearing Algorithm [16], can also be used in implementing tearing decomposition. However, the Tearing Algorithm is just some particular form of the Block LU Factorization [17]. The details of this approach are given in [16].

A series of new circuit simulators, such as SLATE [8] and SAMSON [18], implement tearing decomposition techniques in the solution of the linear algebraic equations and give reasonable improvement. Actually, SAMSON is a mixed mode simulator that also implements a block relaxation technique for solving the nonlinear algebraic equations. We will discuss the relaxation technique later in this chapter.

Multilevel Newton-Raphson techniques [7] can also be used in solving a system of nonlinear algebraic equations decomposed by means of the tearing method. Here we just describe this procedure briefly. The solution strategy has been to estimate the tearing variables (e.g., current or voltage) at each tearing port and to excite the

torn subsystems with independent sources at these ports. The remaining port responses are computed and are substituted into the interconnection equations. If these equations are not satisfied, another guess is made of the variables chosen as port excitations. This iterative procedure continues until convergence is achieved. MACRO [7] is an example of the simulator that implements the multilevel Newton-Raphson method for solving the nonlinear algebraic equations.

2.4 Relaxation Decomposition

In the tearing decomposition approach, the original system of equations may be sparse while the reduced interconnection matrix may not. Therefore, the computational advantage of the tearing decomposition technique over the conventional circuit simulator depends crucially on how small each decomposed subsystem and the reduced interconnection matrix are.

However, in the relaxation decomposition approach, the system of equations is simply partitioned into several subsystems. There is no restriction on the block structure of the system. Within each subsystem, the variables to be solved for are defined as internal variables and the other variables are defined as external variables.

The solution strategy of the relaxation decomposition approach is to solve each subsystem individually, and iterate through all the subsystems until the convergence is achieved. In order to solve a subsystem for its internal variables, the subsystem has to be decoupled by replacing the values of its external variables. Usually, it

takes a number of iterations, repeatedly solving each of the decoupled subsystems, so that the values of the external variables of each subsystem can be updated.

There are two well-known types of relaxation techniques, namely the Gauss-Jacobi (GJ) relaxation [3] and the Gauss-Seidel (GS) relaxation [4]. These approaches as well as the Gauss-Seidel predictor technique [10] will be discussed in detail in Chapter 3. Figure 2.6 shows the use of the relaxation technique at various levels of the system of equations. The relaxation approach applied at the system level of the nonlinear differential equations such as RELAX [9] is the so-called waveform relaxation method which will be discussed in Chapter 4. The relaxation approach applied at the system of algebraic equations such as MOTIS [3], MOTIS-C [4], ,SPLICE [6] and PREMOS [10] is the time-point relaxation method which will be discussed in Chapter 3.

2.5 Concluding Remarks

We have reviewed various decomposition techniques that have been proposed and implemented. The relaxation approach has the potential of achieving the speeds necessary for the next new generation of circuit simulators [11]. However, the relaxation approach does not guarantee that the sequence of the iterated solutions will converge to the exact solution unless the convergence condition on the partitioned system is satisfied.

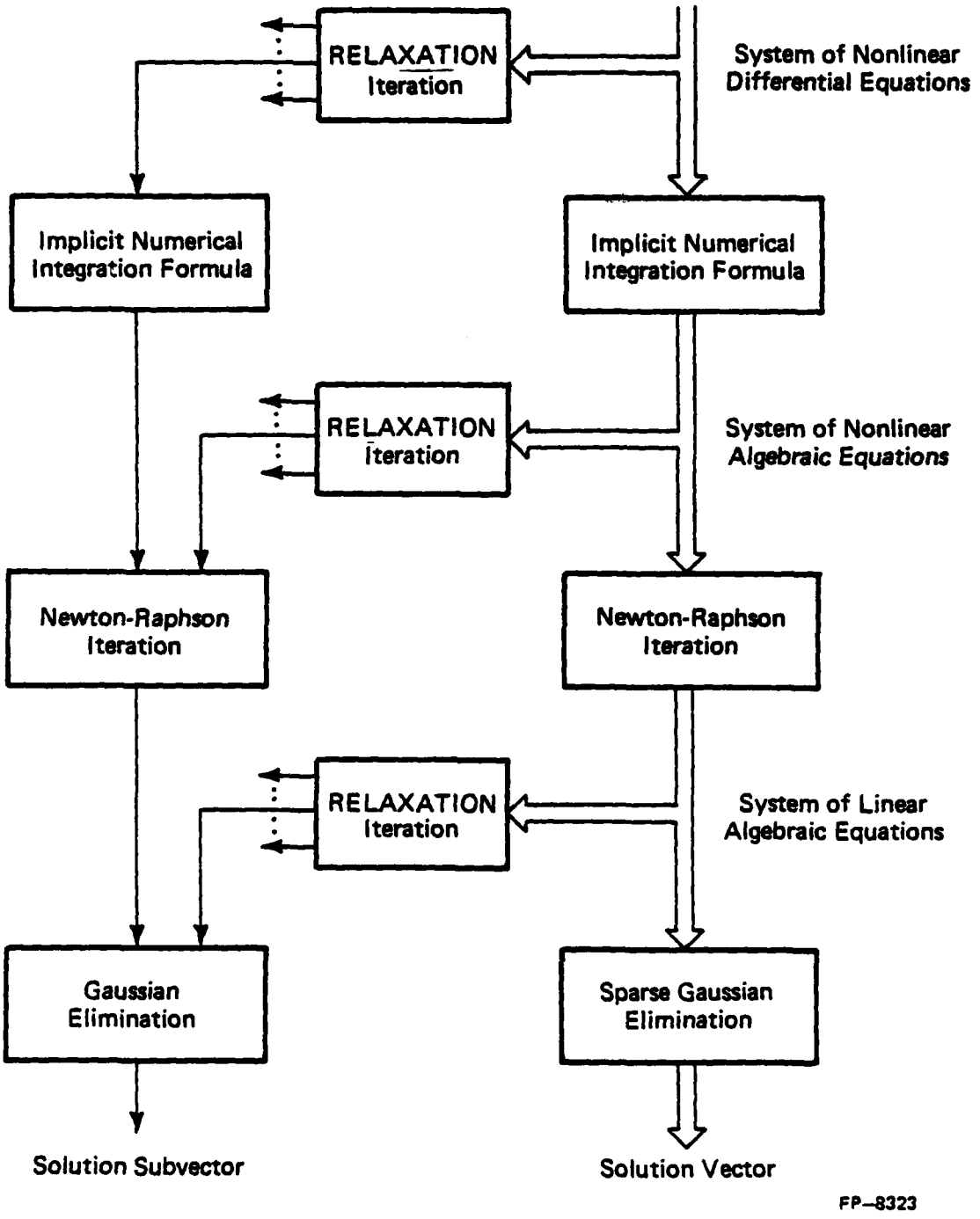


Fig. 2.6 The Use of Relaxation Techniques at Different Levels of System of Equations.

The study of the relaxation technique at different levels of the system of equations is still open. The investigation of the numerical convergence and stability properties of the relaxation techniques is the main theme in this research.

CHAPTER 3

THE MODIFIED GAUSS-SEIDEL METHOD AND ITS NUMERICAL PROPERTIES

In the simulation of integrated circuits , relaxation techniques were proposed for the solution of simultaneous algebraic equations in order to speed up the simulator so that larger circuits could be handled. These methods are iterative and convergence depends on the coupling among the variables. Since many digital MOS circuits have rather weak coupling from output nodes to input nodes because the input is the high impedance gate, these methods should perform well on digital MOS circuits. This reasoning led to the development of the MOTIS program [3], a landmark for the CAD area. Following the MOTIS simulator, a series of the MOTIS-type simulators, such as MOTIS-C [4], SPLICE [6] and PREMOS [10] were developed. The relaxation techniques used in these simulators decompose the system at the level of the difference equation.

In this chapter, we describe the time-point relaxation techniques, such as Gauss-Jacobi relaxation and Gauss-Seidel relaxation, together with a study of their convergence and stability properties. This study reveals why time-point relaxation methods work well on some special classes of circuits, but perform very poorly on other types of circuits.

3.1 Mathematical Formulation

The systems of the nonlinear differential-algebraic equations which describe the performance of the integrated circuits are generally of the following form :

$$f(x(t),\dot{x}(t),u)=0 \tag{3.1}$$

$$E(x(0)-x_o)=0 \tag{3.2}$$

where $x \in R^p$ is the unknown variables at time t with the given initial value x_o; $\dot{x}$ is the time derivative of x at time t; $u \in R^r$ is the vector of all the inputs and their time derivatives;

$f : R^p \times R^p \times R^r \rightarrow R^p$ is a continuous function; and $E \in R^{nxp}$, $n \leq p$ is a matrix of rank n such that E(x(t)) is the state of the system at time t. Let $\{ t_i ; i = 0,1,\ldots,N \}$ denote a sequence of increasing time points selected by the simulator with $t_o = 0$ and $t_N = T$ where T is the given simulation time interval.

At every time point t_n, Equation (3.1) can be approximated by using an implicit integral formula, such as the backward Euler formula, to generate a set of nonlinear algebraic equations of the following form:

$$g(x^n) = 0 \tag{3.3}$$

In a standard circuit simulator, Equation (3.3) is solved by using the modified Newton's method which may take a number of

iterations to reach the solution. At each iteration, a series of algorithmic procedures, such as function and Jacobian evaluations, LU factorization and sparse matrix solution techniques, are repeated. The linearized equations at each iteration in the Newton's method are of the form:

$$A\,x = b \tag{3.4}$$

3.1.1 Gauss-Jacobi Relaxation Method

To fit the fast growth of VLSI systems, a series of new generation simulators have been proposed which depart radically from the standard algorithmic techniques. One of the principal points of departure from the standard simulation approach is relaxation decomposition. There are two common types of time-point relaxation techniques used in the new generation simulators, namely the Gauss-Jacobi relaxation, e.g., in the MOTIS program [3], and the Gauss-Seidel relaxation, e.g., in the MOTIS-C program [4].

In MOTIS, the components of x^n in Equation (3.3) are obtained one at a time by solving a sequence of scalar equations, i.e., at time t_{n+1}, the k-th component of x^{n+1}, x_k^{n+1}, is obtained by solving the following scalar equation:

$$g_k(x_1^n, x_2^n, \ldots, x_{k-1}^n, x_k, x_{k+1}^n, \ldots, x_m^n) = 0 \tag{3.5}$$

In the solution sequence, for each scalar equation, the previous values are used for all the 'exogenous' variables. This process yields an approximation to the solution which can be substituted into Equation (3.3) to determine if the values satisfy the equation. If not, the process is repeated and hopefully the iterates converge rapidly to the solution.

3.1.2 Gauss-Seidel Relaxation Method

MOTIS-C used the Gauss-Seidel algorithm to achieve a better result than that of MOTIS. The Gauss-Seidel relaxation method is quite similar to the Gauss-Jacobi method, but recently updated values of the solved variable are retained. The solution sequence is to solve the following equation for x_k^{n+1}:

$$g_k(x_1^{n+1}, x_2^{n+1}, \ldots, x_{k-1}^{n+1}, x_k, x_{k+1}^{n}, \ldots, x_m^{n}) = 0 \qquad (3.6)$$

The process is repeated until the sequence converges to the solution or the number of iterations becomes excessive. If each subsystem has only one internal variable, i.e., the x_i's in Equation (3.5) and Equation (3.6) are scalar, the relaxation approaches are said to be done pointwise, that is, point Gauss-Jacobi relaxation method and point Gauss-Seidel relaxation. Otherwise, it is said to be blockwise. From the network point of view, the pointwise relaxation methods are equivalent to decomposing the network at each node, whereas the blockwise relaxation methods decompose the network into subnetworks which may consist of more than one node.

In the Gauss-Seidel method, because the previous values are used for those unsolved variables, it has been found that usually the sequence converges much more rapidly to the solution. Originally MOTIS and MOTIS-C were programmed to only do the first iteration and to control the time step to achieve accuracy. For large-scale circuit analysis, this one-sweep approach was used to save additional computational steps. In order to achieve less error and better convergence for the unsolved variables, PREMOS [10] used a one-sweep Gauss-Seidel technique with prediction and got reasonable improvement in the transient analysis of MOS circuits. This technique is described below.

3.1.3 Modified Gauss-Seidel Relaxation Method

The solution strategy of the modified Gauss-Seidel method has been to use a forward first-order linear predictor for the unsolved variables. That is, at the i-th scalar equation, to solve for x_i, the values for the unsolved variables x_j's, $j > i$ are predicted according to the following formula:

$$x^{n+1}_{j,guess} = x^n_j + h_n\left(\frac{x^n_j - x^{n-1}_j}{h_{n-1}}\right) \tag{3.7}$$

Experience has shown that of the above time-point relaxation methods, the Gauss-Seidel method with prediction, used in the program PREMOS, usually performs the best for h small enough. However, the fact that all three methods take only one sweep has proven to cause

accuracy problems in some circuits. Thus more recent versions of MOTIS [19] and SPLICE1 [20] iterate until the sequence converges to a solution. If the number of iterations becomes excessive, the time step is reduced to improve convergence. However, this approach can become computationally inefficient under certain conditions. To understand why, the convergence and stability properties of these time-point relaxation methods are analyzed in the next section.

3.2 The Convergence and Stability Properties

In order to study the stability properties of numerical integration methods, a simple first-order differential equation of the form

$$\dot{x} = a\,x \tag{3.8}$$

is chosen by numerical analysts as a test vehicle. Similarly, in order to study the numerical properties of relaxation methods used to decouple a system of differential equations describing the behavior of digital circuits, the simple linear test circuit in Figure 3.1 was chosen [21]. This test circuit was generated by linearizing the model of a cascade of two inverters in which the transistors are assumed to be active. The elements i_s, g_1 and C_1 represent the Norton equivalent circuit at the output of the first inverter, and the capacitor C_c represents the coupling (feedback) from the output node of the second inverter to its input node #1. The elements C_1, C_2, g_1 and g_2 are all scaled to 1, and g_m and C_c are adjustable.

The test circuit has two natural frequencies on the negative real axis, and their ratio (degree of stiffness) can be increased by increasing C_c or g_m as shown in Table 3.1.

Table 3.1
Effect of C_c and g_m on the time constants of the test circuit

case	C_c(F)	g_m(S)	$\tau_1=-1/\lambda_1$(s)	$\tau_2=-1/\lambda_2$(s)	λ_2/λ_1
1	0.01	1	1.1	0.92	1.22
2	0.10	1	1.5	0.80	1.88
3	1.00	1	4.3	0.70	6.17
4	0.01	10	1.4	0.74	1.87
5	0.10	10	2.8	0.43	6.37
6	1.00	10	13.8	0.22	63.30
7	0.01	100	.6	0.39	6.80
8	0.10	100	12.1	0.099	122.00
9	1.00	100	104.0	0.029	3597.00
10	0.01	1000	11.9	0.085	140.00
11	0.10	1000	102.0	0.012	8696.00
12	1.00	1000	1004.0	0.0030	335570.00

The transient solution to this test circuit is of the form

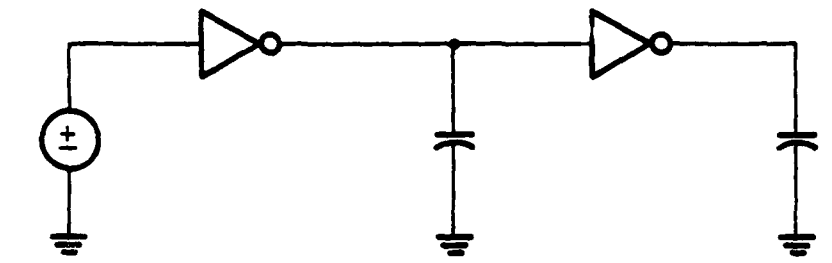
$$v_2(t) = k_1 e^{-t/\tau_1} + k_2 e^{-t/\tau_2} \qquad (3.9)$$

The dominant natural frequency is λ_1, and it determines how slowly the transient response decays. One can estimate the dominant natural frequency using the Miller effect approximation. In the next section, we examine the Gauss-Seidel relaxation method and determine the relation between the stiffness of the test circuit and the time step h needed in order to achieve good convergence to the solution.

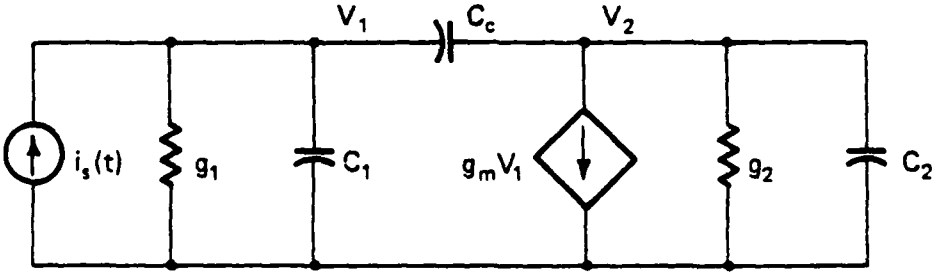
3.2.1 Convergence of the Time-Point Gauss-Seidel Iteration Method

The numerical solution is begun in the usual way, that is, the differential-algebraic Equations (3.1) are converted to a set of algebraic Equations (3.3) by means of an implicit integration formula. The equivalent circuit resulting from this transformation is obtained by replacing the energy storage elements with their companion models. For instance, the node equation of the test circuit of Figure 3.1 can be expressed in the following form:

$$\begin{bmatrix} C_1+C_c & -C_c \\ C_c & C_2+C_c \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} g_1 & 0 \\ g_m & g_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0 \qquad (3.10)$$



(a)



(b)

FP-8322

Fig. 3.1 Test Circuit.

By using the backward Euler formula as follows:

$$\dot{x}_n = \frac{x_n - x_{n-1}}{h_n} \tag{3.11}$$

where h_n is the size of the time step, the equivalent circuit is as shown in Figure 3.2. The equations for this circuit are

$$\frac{1}{h_n}\begin{bmatrix} C_1+C_c+h_n g_1 & -C_c \\ -C_c + h_n g_m & h_n g_2+C_2+C_c \end{bmatrix}\begin{bmatrix} v_{1,n} \\ v_{2,n} \end{bmatrix}$$

$$-\frac{1}{h_n}\begin{bmatrix} C_1+C_c & -C_c \\ -C_c & C_2+C_c \end{bmatrix}\begin{bmatrix} v_{1,n-1} \\ v_{2,n-1} \end{bmatrix} = \begin{bmatrix} i_s(t_n) \\ 0 \end{bmatrix} \tag{3.12}$$

Note that if $C_c = 0$, the two nodes are decoupled. Thus we can solve for v_1 first (the output of the first gate), and then we can solve for v_2 (the output of the second gate) without inverting the circuit matrix. In special purpose circuit simulators for large digital circuits, this is essentially what is done, except on a much larger scale.

If $C_c \neq 0$, the equations (gates) can be decoupled by relaxing certain nodes by means of the Gauss-Seidel method. For instance, for the test circuit, we express Equation (3.12) in the form

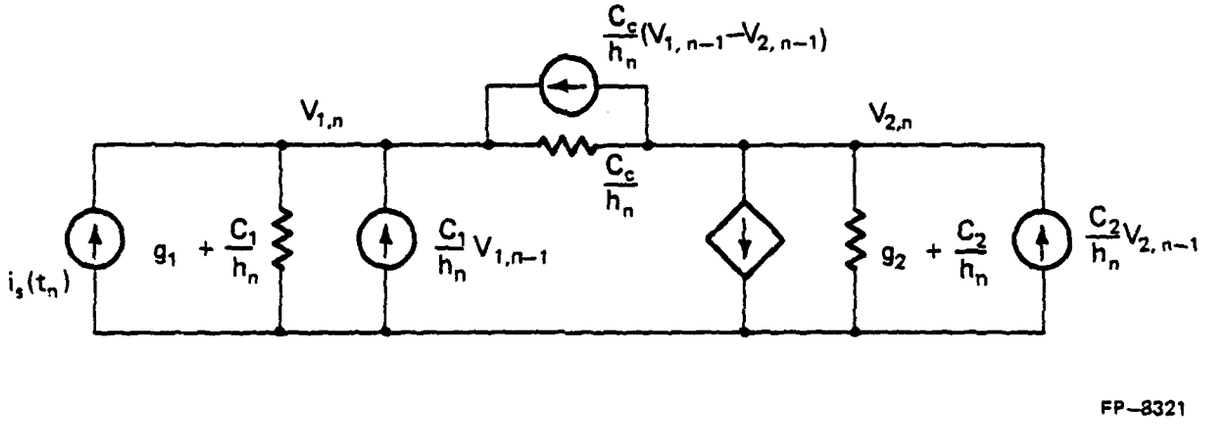


Fig. 3.2 Companion Circuit Model.

$$(L + D) \underline{v}_n = - U \underline{v}_n + B \underline{v}_{n-1} + \underline{i}_s \tag{3.13}$$

where

$$L + D = \frac{1}{h_n} \begin{bmatrix} C_1+C_c+h_n g_1 & 0 \\ -C_c + h_n g_m & h_n g_2+C_2+C_c \end{bmatrix} \tag{3.14}$$

and

$$U = \begin{bmatrix} 0 & -C_c/h_n \\ 0 & 0 \end{bmatrix} \tag{3.15}$$

The Gauss-Seidel method of solution for Equation (3.13) is the iteration

$$\underline{v}_n^{(k)} = -(L+D)^{-1} U \underline{v}_n^{(k-1)} + B \underline{v}_{n-1} + \underline{i}_s \tag{3.16}$$

where

$$\underline{v}_n^{(0)} = \underline{v}_{n-1} \tag{3.17}$$

and

$$-(L+D)^{-1}U = \frac{C_c}{(C_1+C_c+g_1 h_n)(C_2+C_c+g_2 h_n)} \begin{bmatrix} 0 & g_2 h_n+C_2+C_c \\ 0 & C_c-g_m h_n \end{bmatrix} \tag{3.18}$$

No matrix inversion is required, since L+D is triangular. A sufficient condition for these iterates to converge to the solution is

given by the following inequality:

$$||(L+D)^{-1}U|| < 1 \qquad (3.19)$$

The above norm is a function of the step size h_n. Using the l_∞ norm, we found the upper bound on the step size h_n for the range of parameter values in Table 3.1 such that Equation (3.19) is satisfied. This upper bound, h_m, is given in Table 3.2.

Table 3.2
Upper bounds on the time step with the Gauss-Seidel methods

case	C_c(F)	g_m(S)	τ_1	τ_2	h_m	$h_{unstable}$
1	0.01	1	1.1	0.92	*	*
2	0.10	1	1.5	0.80	*	*
3	1.00	1	4.3	0.70	*	*
4	0.01	10	1.4	0.74	*	*
5	0.10	10	2.8	0.43	*	*
6	1.00	10	13.8	0.22	1.0	2.0
7	0.01	100	2.6	0.39	*	*
8	0.10	100	12.1	0.099	0.16	0.32
9	1.00	100	104.0	0.029	0.052	0.104
10	0.01	1000	11.9	0.085	0.14	0.26
11	0.10	1000	102.0	0.012	0.012	0.025
12	1.00	1000	1004.0	0.0030	0.0050	0.010

*Criterion satisfied for all h > 0

The term $h_{unstable}$ is the step size at which the solution becomes unstable in the modified Gauss-Seidel method which will be discussed in the next section.

From Table 3.2 we found that in order to satisfy inequality (3.19) the step size h has to be approximately less than or equal to the size of the smallest time constant in the system. For example, for C_c = 0.1 F and g_m = 100 S, then h < 0.16 s is sufficient for convergence. If g_m is increased to 1000 S, then h < 0.013 s is sufficient for convergence. Our numerical experiments for this circuit concur with the above observations. For instance, for C_c = 0.1 F and g_m =100 S, if h = 0.1 s the iterative solution sequences converged, but they did not converge for h = 0.2 s.

3.2.2 Numerical Stability of the One-sweep Gauss-Seidel Iteration Method with Prediction

In our test circuit we used the following predictor

$$v_{2,n}^{(0)} = v_{2,n-1} + h_n \left(\frac{v_{2,n-1} - v_{2,n-2}}{h_{n-1}}\right) \qquad (3.20)$$

in order to initialize Equation (3.16). If k = 1 in Equation (3.16), then upon substitution of Equation (3.20) into Equation (3.16) we obtain the difference equation:

$$\begin{bmatrix} C_1+C_c + h_n g_1 & 0 \\ (-C_c) + g_m h_n & h_n g_2 + C_2+C_c \end{bmatrix} \underline{y}_n + \begin{bmatrix} -(C_1+C_c) & -C_c \frac{h_n}{h_{n-1}} \\ C_c & -(C_2+C_c) \end{bmatrix} \underline{y}_{n-1}$$

$$+ \frac{h_n}{h_{n-1}} \begin{bmatrix} 0 & C_c \\ 0 & 0 \end{bmatrix} \underline{y}_{n-2} = h_n \begin{bmatrix} i_s(t_n) \\ 0 \end{bmatrix} \qquad (3.21)$$

Applying the Z-transform yields the following characteristic polynomial:

$$Pz^3 + Qz^2 + Rz + S = 0 \qquad (3.22)$$

where for $h_n = h_{n-1} = h$

$$P = (g_1 h + C_1 + C_c)(g_2 h + C_2 + C_c)$$

$$Q = -(C_2 + C_c)(C_1 + C_c + g_1 h) - (C_1 + C_c)(C_2 + C_c + g_2 h)$$

$$+C_c(g_m h - C_c)$$

$$R = (C_1 + C_c)(C_2 + C_c) + C_c^2 + C_c(C_c - g_m h)$$

$$S = -C_c^2$$

The term $h_{unstable}$ represents the time step at which Equation (3.22) has a root on the unit circle. For our test circuit the values of $h_{unstable}$ are given in Table 3.2. If the time step becomes greater than $h_{unstable}$ the integration becomes numerically unstable.

Furthermore, the root loci in Figure 3.3 and Figure 3.4 give more details about the stability of the modified Gauss-Seidel method for the stiff case. The roots of the characteristic equation (3.22) are a function of h. In order for the modified Gauss-Seidel method to be numerically stable, the roots of Equation (3.22) must lie inside the unit circle. For h_n small enough, this is the case. However, we found from the root locus that if h_n is much larger than the smallest time constant in the circuit (approximately a factor of two), these roots lie outside the unit circle. For example, the root locus of the case when C_c = 0.1 F and g_m = 1000 S is shown in Figure 3.3. Note that for h > 0.025 s one of the roots lies outside the unit circle. From Table 3.1 this is approximately $2\tau_2$. Thus, in this case we are constrained by the numerical stability properties of the algorithm to keep the step size in the neighborhood of the smallest time constant over the entire time interval.

Table 3.3 demonstrates the constraint condition of the modified Gauss-Seidel algorithm. The analysis time interval in Table 3.3 is $3\tau_1$, where τ_1 is the larger time constant of the system in each case, such that the system can reach the steady state during the analysis time interval. With the local truncation error 0.001, the number of the time points needed for each case is counted in Table 3.3 to see the effect of the system stiffness on the time step. It is found that the modified Gauss-Seidel method takes many time points for the stiff systems. For example, in case 9 with C_c=1.0 F and g_m=100 S, τ_1 = 104.0 s, it takes 2650 time points in $3\tau_1$; the average time step in

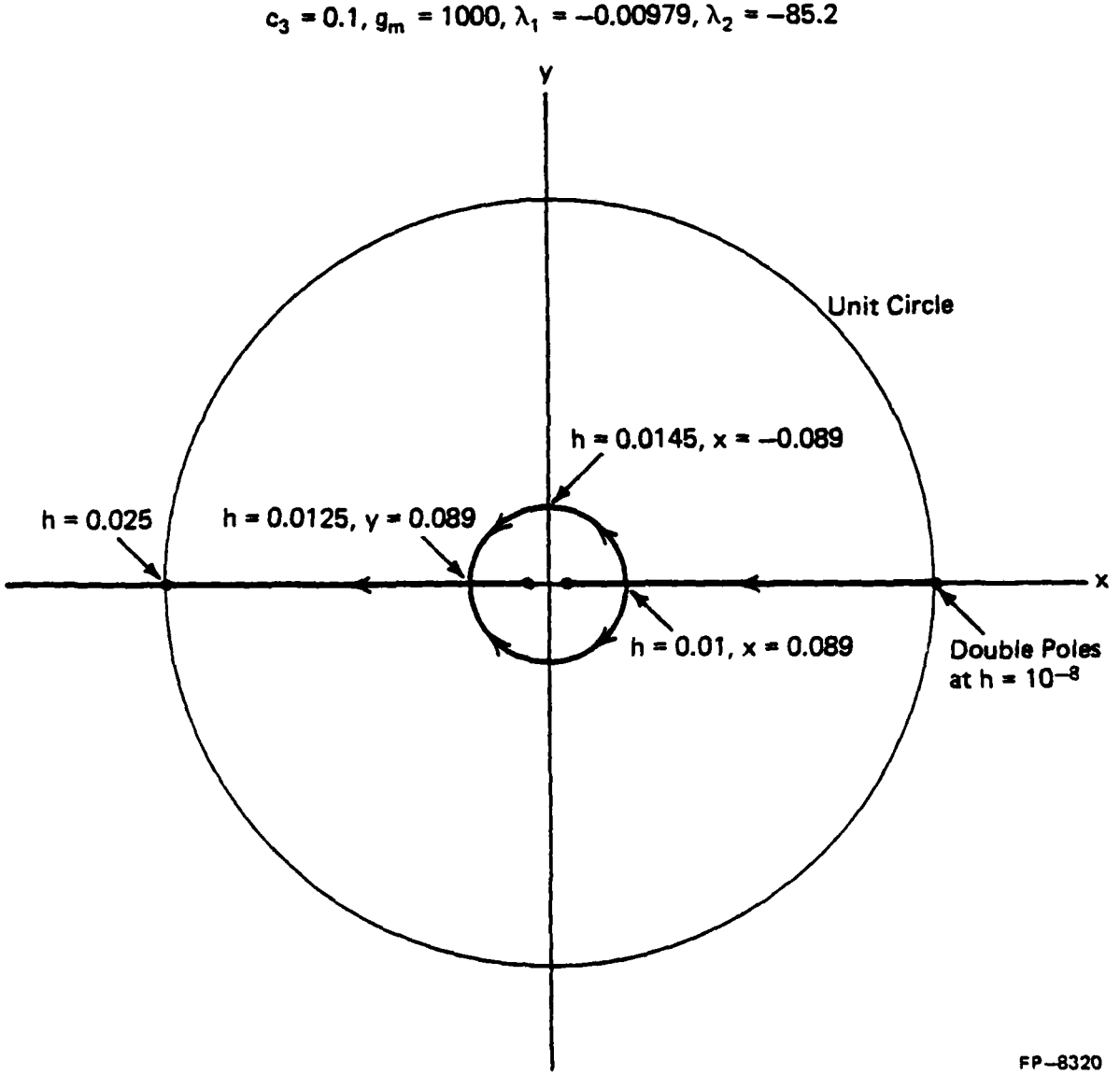


Fig. 3.3 Root Locus of One-Sweep Gauss-Seidel Method.

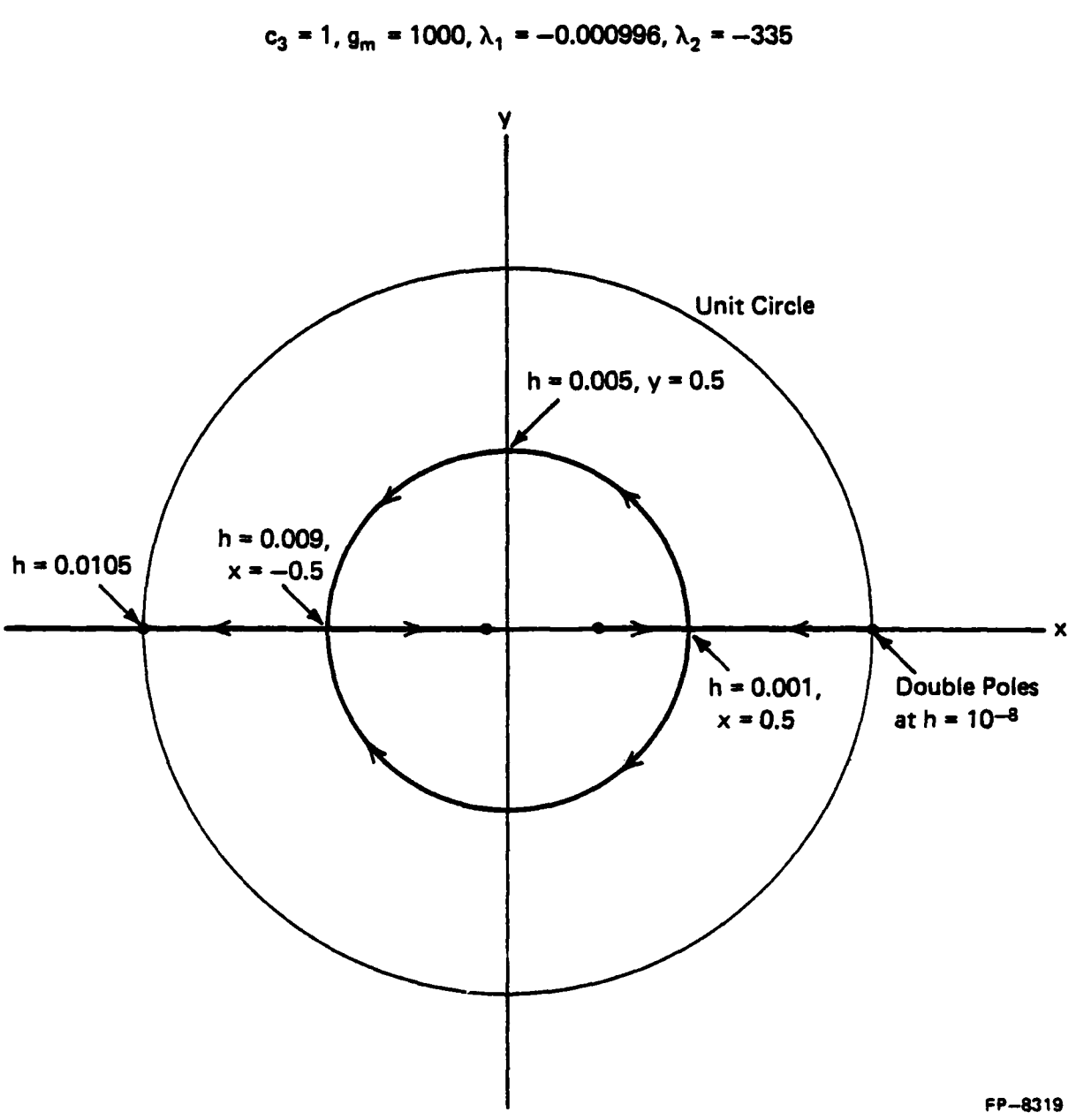


Fig. 3.4 Root Locus of One-Sweep Gauss-Seidel Method with Linear Prediction.

this case is 0.117 s which is about four times the smallest time constant τ_2 (=0.029 s), while for the standard circuit simulation technique it takes only 20 time points with the average time step 15.6 s. Another instance is in case 11 with C_c=0.1 F and g_m=1000 S, τ_1 = 102.0 s, and it takes 8122 time points in $3\tau_1$, the average time step in this case is 0.037 s which is about three times the smallest time constant τ_2 (=0.012 s), while for standard circuit simulation it takes only 44 time points with the average time step 6.96 s.

Table 3.3.
The number of time points for variable time step (LTE= .001)

case	λ_2/λ_1	modified Gauss-Seidel	standard method
1	1.22	5	6
2	1.88	6	6
3	6.17	8	7
4	1.87	6	7
5	6.37	9	9
6	63.30	7	6
7	6.80	17	17
8	122.00	91	19
9	3597.00	2650	20
10	140.00	231	45
11	8689.00	8122	44
12	335570.00	90048	45

However, for those nonstiff systems such as cases 1 to 7, both the modified Gauss-Seidel method and the standard circuit simulation technique take the same number of time points. It is clear from Table 3.3 that, for stiff systems, the time steps have to be constrained to the order of the smallest time constants of the system to keep the solutions numerically stable.

3.3 Concluding Remarks

3.3.1 Remark I

In the above tables, the parameters $C_1 = 1$, $C_2 = 1$, $g_1 = 1$ and $g_2 = 1$, the floating capacitor C_c and the transconductance g_m are varied to change the stiffness of the system. If $C_c = 0$, the system in Figure 3.1 contains two separated subcircuits. The circuit becomes coupled with the introduction of the capacitor C_c. If the coupling is too strong, the system becomes stiff and the modified Gauss-Seidel method does not work well as seen in Table 3.3.

The conclusion of the numerical properties of the time-point Gauss-Seidel with (or without) prediction drawn from this linear test circuit is that [21,22], if the coupling element significantly affects the natural frequencies of the circuit and creates a stiff system, then no advantage is gained by using implicit integration methods because the convergence of the Gauss-Seidel iterates requires that the step size be approximately no larger than the smallest time constant in the circuit over the entire time interval. The

computational cost can be significantly increased by this constraint, so much so that all the advantages gained by decoupling the circuits cannot only be lost, but the computational cost might even exceed that of a general purpose circuit simulator.

Figure 3.3 and Figure 3.4 show that the modified Gauss-Seidel technique goes numerically unstable when the time step exceeds the smallest time constant of the system by a factor of three or four. Given the above results, one would expect the modified Gauss-Seidel method to perform poorly in the transient analysis of MOS circuits in which the floating gate-drain capacitance is modeled. In such circuits the pole-splitting phenomenon [16] occurs when the transistors are active. The degree to which the poles split determines the stiffness of the system, and is determined by the low frequency gain of the logic gate, e.g., slope of the dc characteristic in the active region. However, due to the gate delay and the nonlinear characteristics of the MOS transistor, the pole-splitting phenomenon does not seem to be very strong [10]. Figure 3.5 shows that the modified Gauss-Seidel method works well for the 3-stage ring oscillator.

3.3.2 Remark II

In the other case, suppose there is a coupling capacitor between two subnetworks whose time constants are already widely separated, that is, in Figure 3.1 suppose $g_1 = 1.0$, $C_1 = 1$, and $C_2 = 1$, $g_2 = 1.0\text{x}10^{-4}$. Table 3.4 shows the required number of time points with a variable time step when the transconductance $g_m = 1$ and the floating

capacitor C_c is varied.

Table 3.4
The number of time points for variable time step (LTE= .00001)

case	C_c(F)	λ_2/λ_1	method I	method II	method III
1	0.00	10000	570	570	570
2	0.01	10200	570	574	570
3	0.10	12000	569	608	570
4	1.00	30000	397	765	570

where

method I : Modified Gauss-Seidel Technique
method II : Gauss-Seidel Technique
method III : Standard Circuit Simulation

Table 3.4 shows the two decoupled subsystems (C_c) are already stiff and the coupling capacitor C_c does not significantly affect the time constants. For example, with $C_c = 0$ one subsystem has a time constant equal to 1 s, and the other has a time constant equal to 10000 s. Note in this example the number of time points for each case in Table 3.4 does not vary significantly as the coupling capacitor changes. Contrary to the results shown in Table 3.3, in this case the coupling does not affect the convergence and stability properties too much. This conclusion also concurs with some other researcher's observations [23].

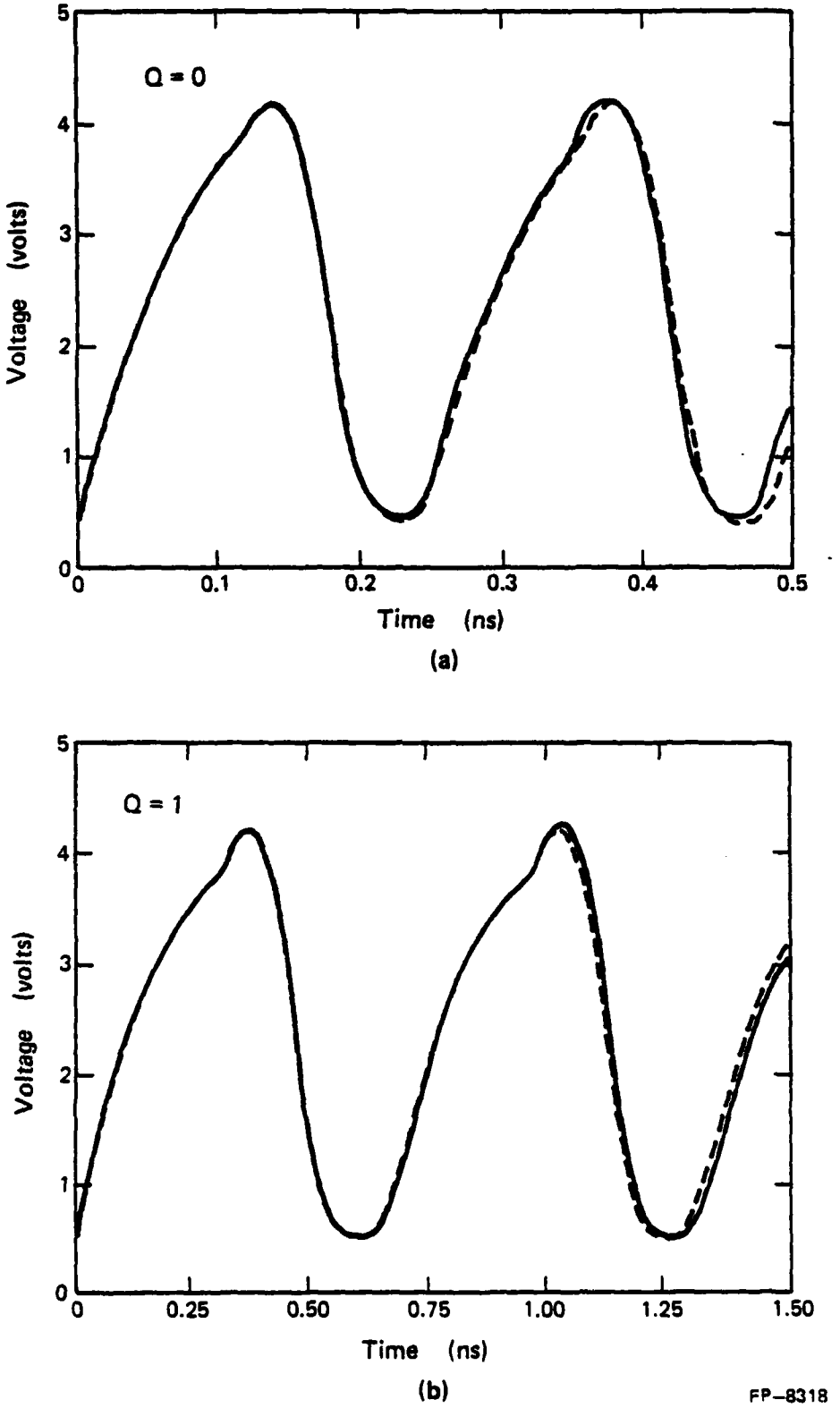


Fig. 3.5 Applied the Modified Gauss-Seidel Method to the 3-Stage Ring Oscillator.
(Solid Line Is the Exact Solution, $Q = C_c/C_1$.)

In [23], Gear studied the simulation of an aircraft which was simplified to the two-subsystem model as shown in Figure 3.6.

The control subsystem reacts very rapidly (being electronic) whereas the flight dynamic subsystem reacts relatively slowly, being a mechanical change. Because the dynamics are slow, his conclusion was that, there can be very little coupling from the control to the dynamics, and one can break the feedback loop from the dynamics to the control and handle each subsystem separately. Here we keep $C_1 = C_2 = 1$, by adjusting $g_1 = 1$ and $g_2 = 1.0 \times 10^{-4}$ to create the similar environment as in [23] and achieve the same conclusion.

In the next chapter, the waveform relaxation method and its convergence and stability properties will be discussed.

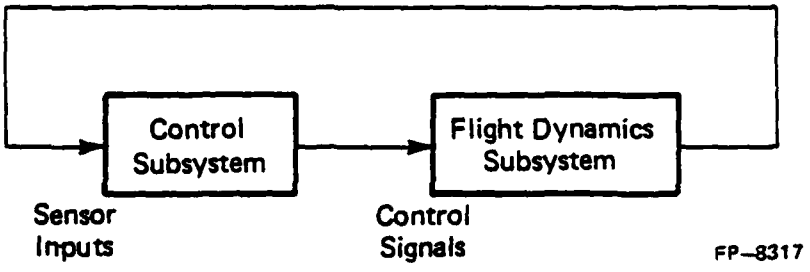


Fig. 3.6 Simplified Simulation Model.

CHAPTER 4

THE WAVEFORM RELAXATION METHOD AND ITS NUMERICAL PROPERTIES

In this chapter, we will discuss a second avenue by which the relaxation of nonlinear systems has been approached. While the first approach, presented in Chapter 3, decomposes the system into several subsystems at the level of the difference equations, the second decomposes the system at the level of the ordinary differential equations. This alternative relaxation method at the differential equation level must deal with elements in function spaces, i.e., waveforms. Thus, it is classified as the waveform relaxation method. This approach began with the work which led to the RELAX program [9]. Either the Gauss-Jacobi method or the Gauss-Seidel method could be used in the waveform relaxation algorithm.

A brief mathematic description of the waveform relaxation method together with its convergence and stability properties will be discussed in this chapter.

4.1 Mathematical Formulation

Let us recall the set of algebraic-differential equations in Equation (3.1) and Equation (3.2) as follows:

$$f(x(t),\dot{x}(t),u)=0 \tag{4.1}$$

$$E(x(0)-x_0)=0 \tag{4.2}$$

where $x \in R^p$ is the unknown variable at time t with the given initial value x_0; $\dot{x}$ is the time derivative of x at time t; $u \in R^r$ is the vector of all the inputs and their time derivatives;

$f : R^p \times R^p \times R^r \rightarrow R^p$ is a continuous function, and $E \in R^{n \times p}$, $n \leq p$ is a matrix of rank n such that E(x(t)) is the state of the system at time t. Let $\{ t_i ; i = 0,1,\ldots,N \}$ denote a sequence of increasing time points selected by the simulator with $t_0 = 0$ and $t_N = T$, where T is the given simulation time interval.

The general frame of the waveform relaxation algorithm consists of two major processes, namely, the assignment-partition process and the relaxation process. The dynamic system is decoupled into certain subsystems through the first process, while the second one yields the waveform of each subsystem at each iteration.

4.1.1 The Assignment-Partitioning Process

In this section, we describe the first process of the waveform relaxation method, that is, the assignment-partitioning process, to decouple the system of nonlinear algebraic-differential equations into subsystems. In the assignment-partition process, each unknown variable is assigned to an equation of (4.1) in which it is involved. However, no two variables can be assigned to the same equation; therefore, Equation (4.1) is partitioned into m disjoint subsystems as follows :

$$f_1(\dot{x}_1, x_1, d_1, u) = 0$$
$$\vdots$$
$$f_m(\dot{x}_m, x_m, d_m, u) = 0 \qquad (4.3)$$

$$E(x(0) - x_o) = 0 \qquad (4.4)$$

where for each $i = 1, 2, \ldots, m$ $x_i \in R^{p_i}$ is the subvector of unknown variables assigned to the i-th partitioned subsystem and

$$d_i = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m, \dot{x}_1, \ldots, \dot{x}_{i-1}, \dot{x}_{i+1}, \ldots, \dot{x}_m)^T \qquad (4.5)$$

In the i-th subsystem f_i, x_i and x_j, $i \neq j$ are called vector endogenous and exogenous variables, respectively. If we treat the vectors d_i, $i = 1, 2, \ldots, m$ as inputs, then (4.3) can be solved by solving m independent subsystems. In other words, the system has been decoupled into m subsystems, and d_i's are called the decoupling vectors of the system. The second process, relaxation process, will be introduced in the next section.

4.1.2 The Relaxation Process

The relaxation process starts with an initial guess of the waveform solutions of (4.1) in order to initialize the approximated

waveforms of the decoupling vectors. It's an iteration process; during each iteration, each decomposed subsystem x_i is solved for its endogenous variables in the entire time evolution [0, T] by using the approximated waveform of its decoupling vector. The iterative process is carried out repeatedly until satisfactory convergence is achieved.

The actual implementation of the Waveform relaxation algorithm can be described as follows:

1) Assignment-partition process

Assign the unknown variables to equations in Equation (4.1) and partition Equation (4.1) into m subsystems of equations as by Equation (4.3).

2) Initialization of the relaxation process

Set k = 1 and guess an initial waveform ($x^0(t)$; $t \in [0, T]$); the typical guess is $x^0(t) = x(0)$ for all $t \in [0, T]$.

3) Analysis of the decomposed system at the k-th iteration

i) For the Gauss-Jacobi Waveform Relaxation method

for each i= i, 2, .., m, set

$$d_i^k = (x_1^{k-1}, \ldots, x_{i-1}^{k-1}, x_{i+1}^{k-1}, \ldots, x_m^{k-1}, \dot{x}_1^{k-1}, \ldots, \dot{x}_{i-1}^{k-1}, \dot{x}_{i+1}^{k-1} \ldots, \dot{x}_m^{k-1})^T \quad (4.6)$$

ii) For the Gauss-Seidel Waveform Relaxation method

for each i = i, 2, .., m, set

$$d_i^k = (x_1^k, \ldots, x_{i-1}^k, x_{i+1}^{k-1}, \ldots, x_m^{k-1}, \dot{x}_1^k, \ldots, \dot{x}_{i-1}^k, \dot{x}_{i+1}^{k-1}, \ldots, \dot{x}_m^{k-1})^T \quad (4.7)$$

With the decoupling vector d_i's, solve for $(x_i{}^k(t)$; $t \in [0, T])$ from Equation (4.3).

4) Iteration loop

Set k = k + 1 and go to 3). The iteration process stops when the difference between $(x^k(t); t\in[0, T])$ and $(x^{k-1}(t); t\in[0, T])$, i.e., $\max_{t\in[0,T]} \| x_k(t) - x_{k-1}(t) \|$, is sufficiently small.

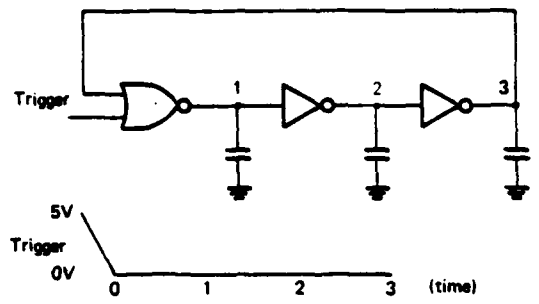
In the next section, we will describe the modified window waveform relaxation method. From a test circuit, it is shown that the modified technique requires less CPU time and memory storage.

4.2 The Modified Waveform Relaxation Method and Its Numerical Properties

At each iteration, the waveform relaxation method decomposes the system into several subsystems each of which is analyzed for the entire given time interval. The accuracy and convergence properties for the waveform relaxation method have been studied in [9]; however, the huge memory storage needed to store the waveforms of each subsystem in the entire time evolution [0, T] can be expensive for large systems.

However, in some circuits the iterates oscillate about the solution. In these cases, at each iteration only a specific part of the waveform in each subsystem can be useful in analyzing other subsystems. Under this condition, sweeping through the entire time evolution [0, T] is somewhat of a waste in either CPU time or memory storage. For example, for a three-stage ring oscillator in Figure 4.1, Figure 4.2 contains the solution waveforms of the circuit after each iteration. For the waveform of the first iteration in Figure 4.2, we can find that the information beyond t_1 is quite different from that of the actual waveform, hence it's meaningless using the waveform beyond t_1 to analyze other subsystems.

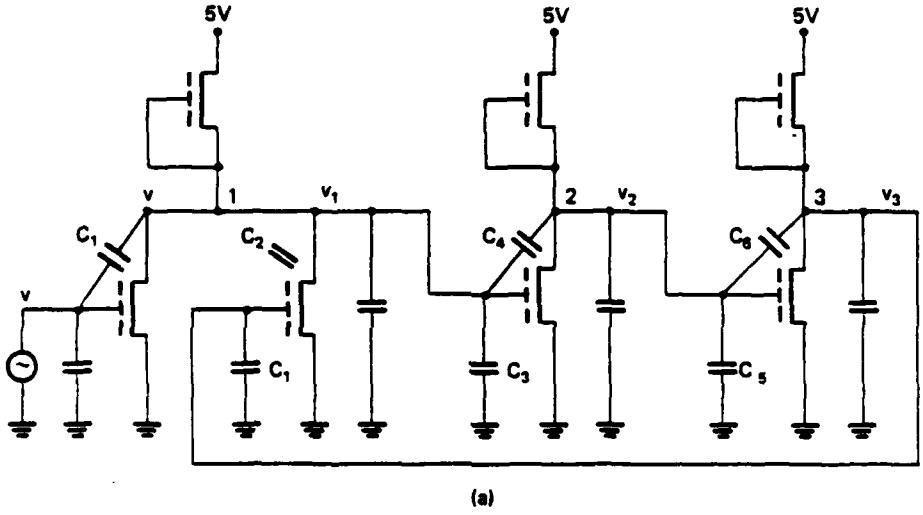
In the next section, we use the circuit in Figure 4.1 as an example to illustrate the basic concept of the modified waveform relaxation algorithm.



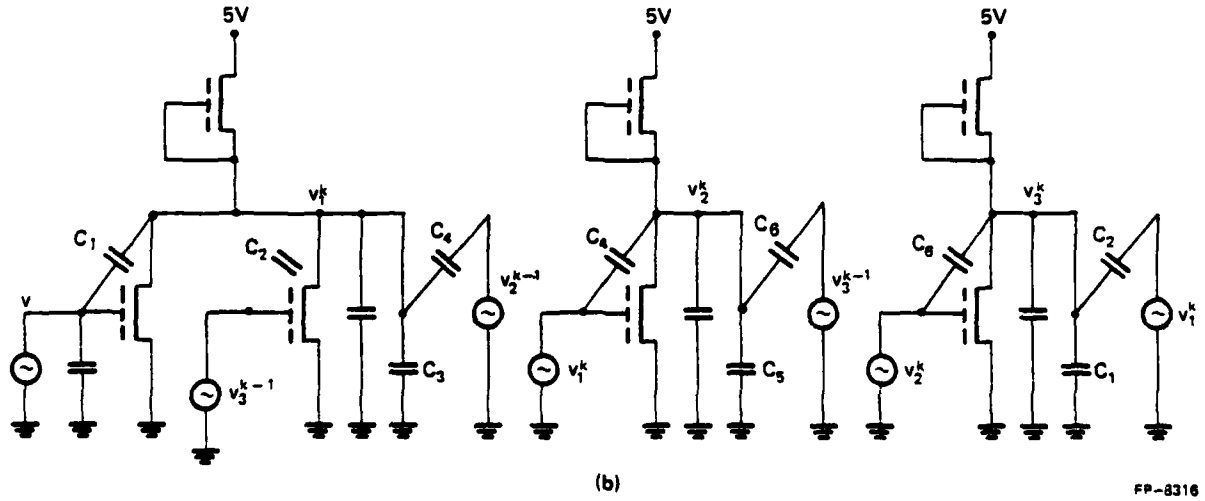
$Q = C_2/C_1$

(a) A MOS Ring Oscillator.

(b) The Circuit Interpretation of Its Decomposed Circuit at The k-th Iteration of The GS-WRM Algorithm.



(a)



(b)

FP-8316

Fig. 4.1 A 3-Stage Ring Oscillator.

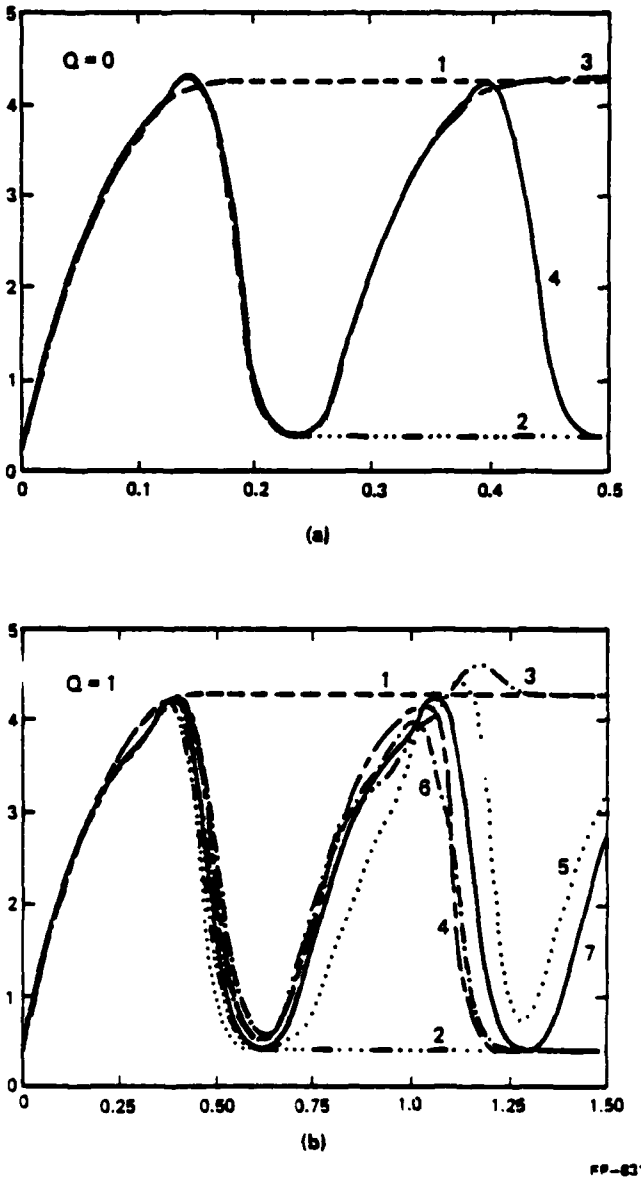


Fig. 4.2 Window Size = T, Solid Line Is the Exact Solution. The Number of Iterations Increases as the Coupling Capacitance Increases.

4.2.1 The Modified Waveform Relaxation Method

For the circuit in Figure 4.1, if we cut the entire time evolution [0, T] into n + 1 time 'windows', i.e., [0, t_1], [t_1, t_2], .., [t_n, T], and, instead of sweeping the iterations in [0, T], we sweep the iterations in each time 'window'. In other words, after reaching the convergent waveform in window 1 (i.e., [0, t_1]) then process the iteration sweep in window 2 (i.e.,[t_1, t_2]), ..., eventually obtaining the waveforms in each time window. Concatenating these waveforms yields the waveform in the entire time evolution [0, T]. This is the basic idea of the modified window waveform relaxation method [12].

The fundamental algorithm of the original Gauss-Seidel Waveform relaxation method in [9] can be described as follows :

The Gauss-Seidel Waveform relaxation algorithm

```
BEGIN
   x = [ Voltages, Currents ]
   n = 1
   WHILE ( δ^n  < Tolerance ) DO
      BEGIN
         FOR subsystem i, i = 1 TO m DO
            BEGIN
               FOR time t = 0 TO t = T DO
                  BEGIN
                     Solve nonlinear equations
```

$$\dot{x}_i^{n+1} = f_i(x_1^{n+1},\ldots,x_{i-1}^{n+1},x_i^{n+1},x_{i+1}^{n},\ldots,x_m^{n})$$

and

$$x_i^{n+1}(0) = x_i^{n}(0) = x_o$$

END

END { sweep m subsystems }

$$\delta^{n+1} = \max_i \max_t \| x^{n+1}(t) - x^{n}(t) \|$$

n = n + 1

END { waveform iteration loop }

END

While the Modified Window Waveform relaxation method is of the following form :

The Modified Window Waveform relaxation algorithm

BEGIN

x = [Voltages, Currents]

w = [Windows]

j = 1

BEGIN

FOR window j, j = 1 TO k DO

BEGIN

FOR subsystem i, i = 1 TO m DO

```
n = 1

   WHILE ( δ^n  < Tolerance ) DO

      BEGIN

         FOR time t = 0 TO t = t_j DO

            BEGIN

               Solve nonlinear equations
```

$$\dot{x}_i^{n+1} = f_i(x_1^{n+1},\ldots,x_{i-1}^{n+1},x_i^{n+1},x_{i+1}^{n},\ldots,x_m^{n})$$

and

$$x_i^{n+1}(0) = x_i^{n}(0) = x_o$$

```
            END

         END { sweep m subsystems }
```

$$\delta^{n+1} = \max_i \max_t \| x^{n+1}(t) - x^{n}(t) \|$$

```
         n = n + 1

      END { waveform iteration loop }

   j = j + 1

END { window process }

END
```

For the ring oscillator circuit, Figure 4.2a and Figure 4.2b show the waveforms after each iteration taking the whole time interval [0, T] as the time window, whereas Figure 4.3a and Figure 4.3b show the waveforms for taking the window size half of the entire time

interval [0, T]. From Figure 4.2a and Figure 4.2b we find that the number of iterations increases as the coupling capacitors increase. In Figure 4.2a, when there is no coupling capacitors, it takes 4 iterations to achieve convergence. In Figure 4.2b with the introducing of the coupling capacitors, it takes 10 iterations to achieve convergence.

Comparing the waveforms in Figure 4.2b with the waveforms in Figure 4.3b, one can find the effect of the window sizes on the waveforms. In Figure 4.2b the converged part of the waveforms increases (propagates) about a half cycle per iteration, while in Figure 4.3b it increases (propagates) about one cycle per iteration and, hence, needs a fewer number of iterations than in Figure 4.2b to achieve convergence for the entire time interval. It is found that in Figure 4.3b it takes 6 iterations to achieve convergence, while it takes 10 iterations to achieve convergence in Figure 4.2b.

During the iterations, the waveforms oscillate about the exact solution. Table 4.1 shows the results for different sizes of the time window under different degrees of coupling capacitance.

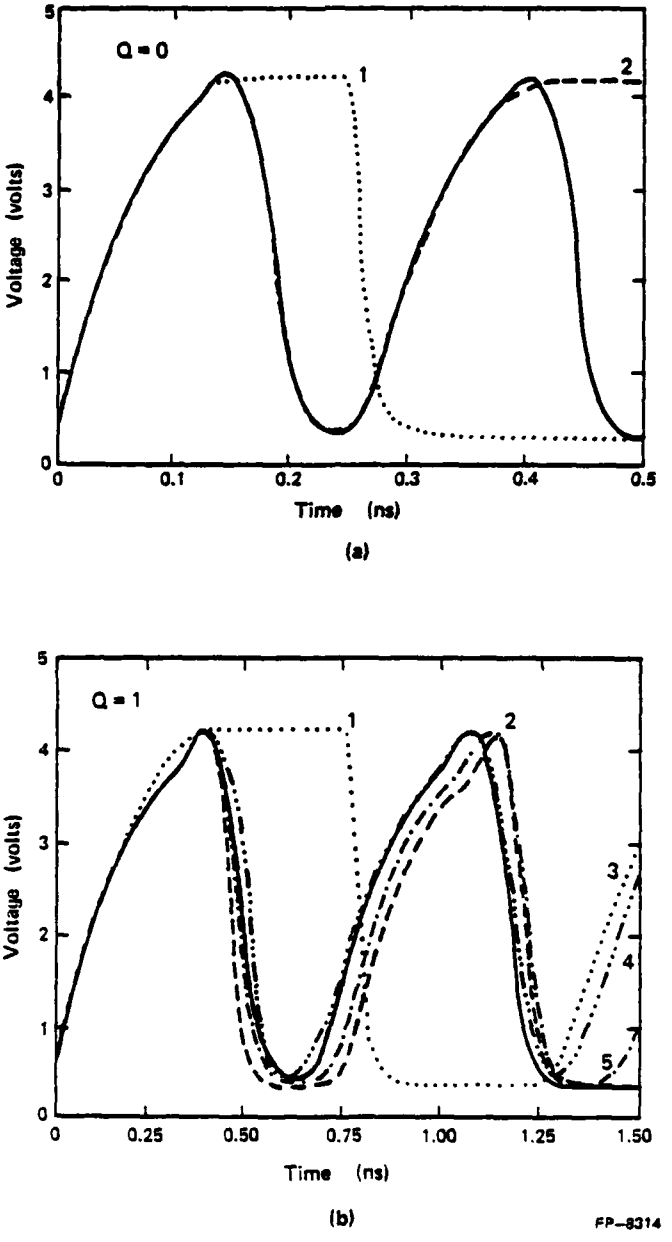


Fig. 4.3 Window Size = T/2, Solid Line Is the Exact Solution. The Number of Iterations Decreases as the Window Size Decreases.

Table 4.1
Number of iterations in each window to reach convergence

window size	Q=0	Q=0.5	Q=1.0	Q=1.5
T	4	6	6	6
T/2	3	5	5	4
T/4	3	3	4	4
T/10	2	3	3	3
T/500	1	3	3	3

* (Q is the ratio of floating capacitance to the ground capacitance)
** Use fixed time step h = T/500 in each window.

It is easy to conclude that the modified window waveform relaxation method achieves more accurate results with less memory storage and less CPU time. The problem that arises with the introduction of the modified waveform relaxation algorithm is how to choose the location and size of the windows. In other words, how should the time interval be cut into several subintervals dynamically? This is an interesting open topic. In the next section we study the convergence properties of the waveform relaxation methods as a function of the coupling and window size.

4.2.2 The Numerical Properties of The Waveform Relaxation Method

Mathematically, the convergence of the waveform relaxation method has been discussed in [9]. For an important class of dynamic systems, MOS digital integrated circuits, it is concluded [9] that for MOS circuits with a grounded capacitor at each node convergence is guaranteed for any arbitrary piecewise continuous set of initial waveforms for the node voltage of the circuit.

In this research, we would like to study the numerical properties of the waveform relaxation algorithm from a different point of view: the effect of the system stiffness on this algorithm. Practically,. in order to investigate the numerical properties of the modified waveform relaxation method , we used the same linear model of the MOS inverter as shown in Figure 3.1 to see the effect of stiffness on the transient analysis of MOS circuits. The implementation of the waveform relaxation algorithm to this linear test circuit is shown in Figure 4.4. In Figure 4.4a, at the (k+1)-th iteration, the node voltage $v_1^{k+1}(t)$ is determined with the node voltage $v_2(t)$ frozen at $v_2^k(t)$ which is the waveform at node #2 at k-th iteration. Once $v_1^{k+1}(t)$ is found, we go to Figure 4.4b and place a grounded voltage source $v_1^{k+1}(t)$ at node #1 and solve for $v_2^{k+1}(t)$, then repeat the iteration until convergence is achieved.

If the backward Euler formula is used, the iterative procedures are described by following equations:

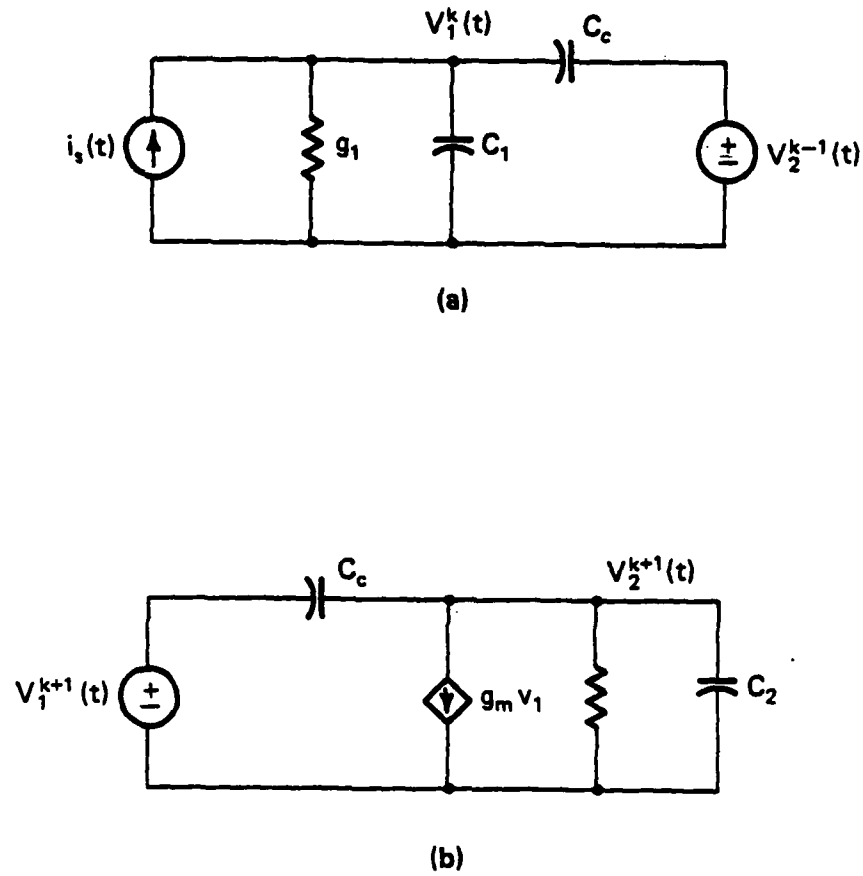


Fig. 4.4 The Implementation of the Waveform Relaxation Method to the Test Circuit in Fig. 3.1.

$$\left(\frac{C_1+C_c}{h_n} + g_1 \right) v_{1,n+1}^{k+1} - \frac{C_1}{h_n} v_{1,n}^{k+1} - \frac{C_c}{h_n} \left(v_{1,n}^{k+1} - v_{2,n}^{k} \right)$$

$$- \frac{C_c}{h_n} v_{2,n+1}^{k} = i_s(t_{n+1}) \qquad (4.8)$$

and

$$\left(g_2 + \frac{C_2+C_c}{h_n} \right) v_{2,n+1}^{k+1} + g_m v_{1,n+1}^{k+1} - \frac{C_2}{h_n} v_{2,n}^{k+1}$$

$$+ \frac{C_c}{h_n} \left(v_{1,n}^{k+1} - v_{2,n}^{k+1} \right) - \frac{C_c}{h_n} v_{1,n+1}^{k+1} = 0 \qquad (4.9)$$

where h_n is the size of the time step and $v_{1,n+1}^{k+1}$ is the voltage of node #1 at the (k+1)-th iteration at (n+1)-th time point. Equation (4.8) and Equation (4.9) are solved iteratively until convergence is achieved.

The modified waveform relaxation method solves (4.8) and (4.9) in each time window and concatenates the results yielding the waveform in the entire time interval. Table 4.2 shows the required number of iterations in each window to get a convergent solution.

Table 4.2
Number of iterations for WRM with h = 0.1

case	C_c	g_m	$\tau_1=-\lambda_1$	$\tau_2=-\lambda_2$	λ_2/λ_1	Node eq	State eq
1	0.01	1.0	1.1	0.92	1.22	1	2
2	0.10	1.0	1.5	0.80	1.88	2	3
3	1.00	1.0	4.3	0.70	6.17	5	4
4	0.01	100.0	2.6	0.39	6.80	5	4
5	0.10	100.0	12.1	0.099	122.00	a*	7
6	0.01	1000.0	11.9	0.0857	140.00	b*	10
7	0.10	1000.0	102.0	0.012	8696.00	c*	10

a*:greater than 300
b*:greater than 600
c*:greater than 1000

The results in Table 4.2 were obtained with a fixed step size h = 0.1 s. However, from Figure 4.5 we find that similar results were obtained with h = 0.01 s and h = 0.001 s, because in stiff systems the convergence rate of the waveform relaxation method depends on the window size, that is, the time increment over which the iteration is performed. Figure 4.6 graphically illustrates the convergence characteristics of the waveform relaxation method for a given stiff system. For example, in the case C_c = 0.01 F and g_m = 1000 S, the convergence rate as a function of the window size is shown in Table 4.3 for a fixed step size h = 0.1 s.

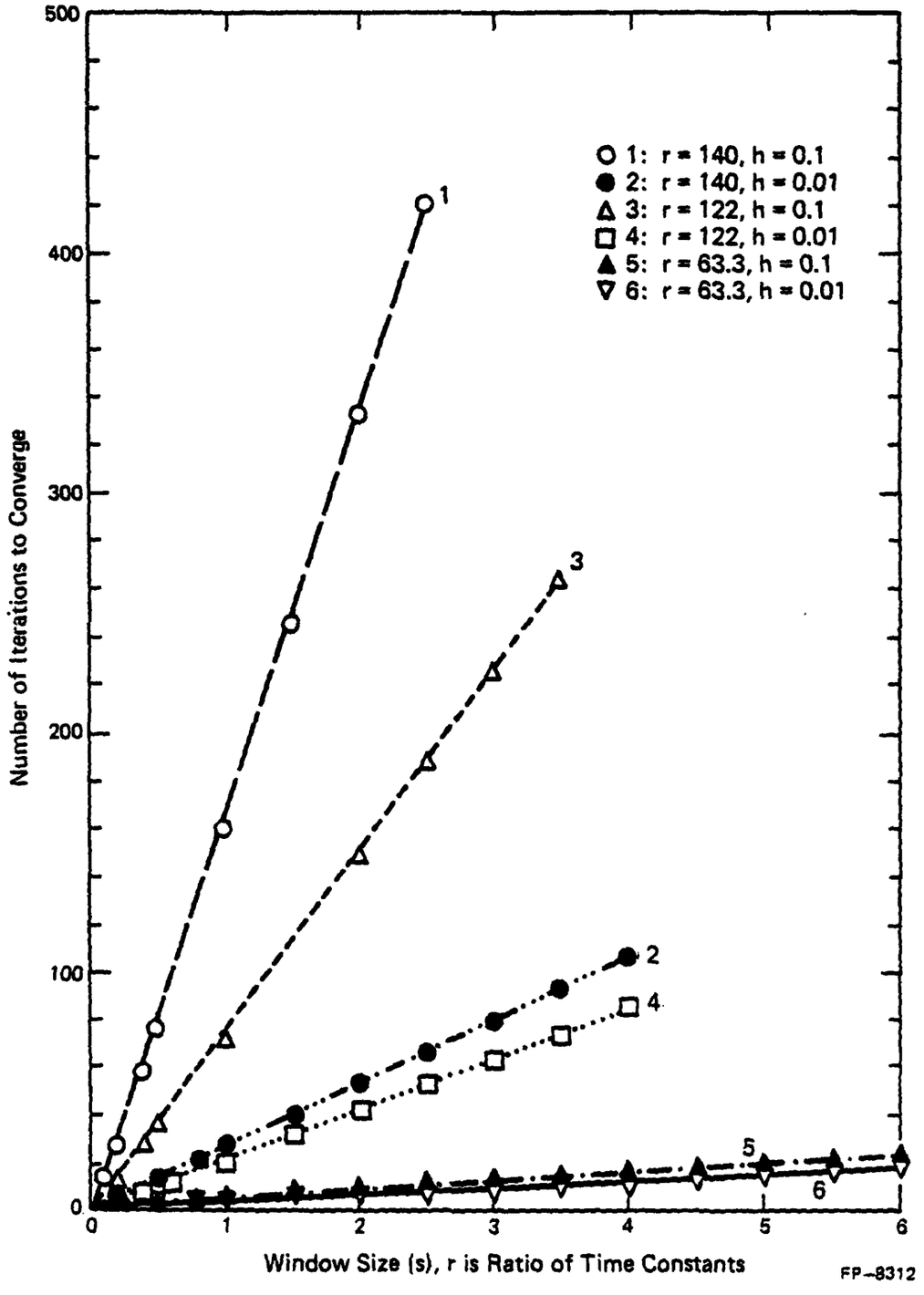


Fig. 4.5 Waveform Relaxation Method Converges as a Function of the Window Size and Time Step.

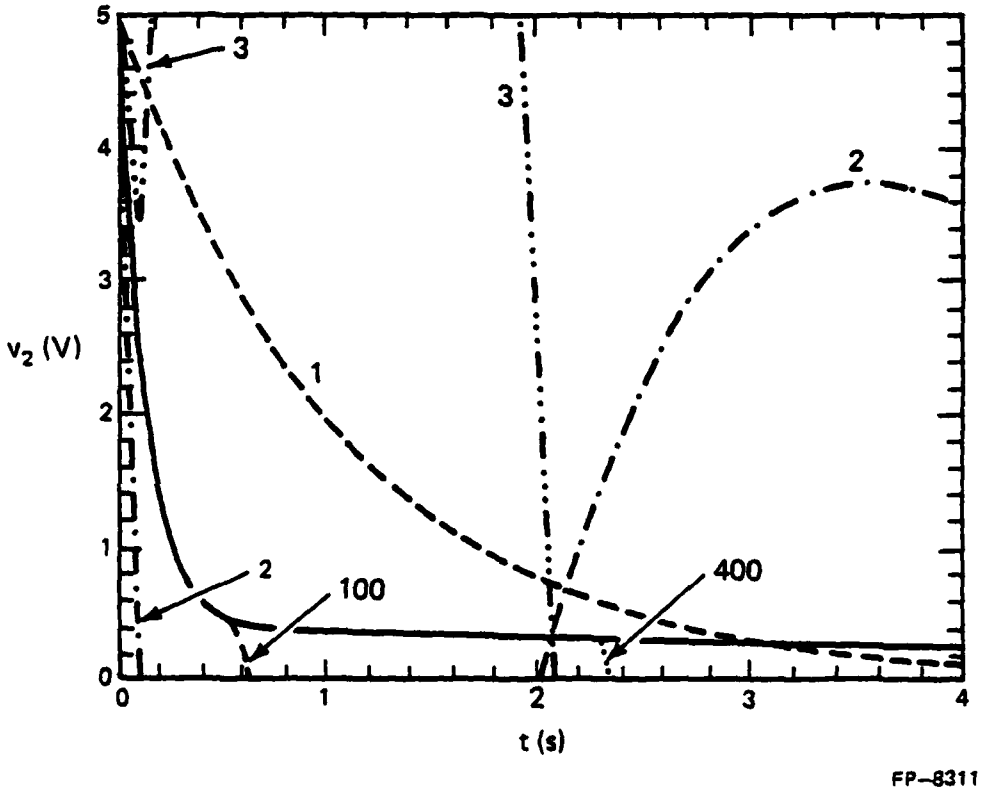


Fig. 4.6 Convergence of the Waveforms with A 4 s Window.

Table 4.3
Convergence as a function of window size

Window size	# of iterations
4s	>600
1s	200
0.5s	90
0.2s	35
0.1s	3

Thus, in the first case the waveforms were relaxed over the entire 4 s interval and over 600 iterations were required for the iterates to converge to the solution. When the waveforms were partitioned into four windows, 200 iterations in each window were required to achieve convergence. Finally, Table 4.3 shows that rapid convergence of the waveform relaxation method was not achieved until the waveforms were partitioned into forty 0.1 s windows. In this case, the window size (0.1 s) is approximately limited to the smallest time constants (0.085 s) to achieve reasonable convergence speed.

4.2.3 The Waveform Relaxation Method Applied to the State Equations

The equations for the test circuit in Figure 3.1 can be rewritten in the following state equation format assuming $i_s(t) = 0$,

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} = A \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{4.10}$$

where

$$A(1,1) = [\ -(C_2 + C_c)g_1 - C_c g_m\ ]/DE$$

$$A(1,2) = [\ -C_c g_2\ ]/DE$$

$$A(2,1) = [\ -(C_1 + C_c)g_m - C_c g_1\ ]/DE$$

$$A(2,2) = [\ -(C_1 + C_c)g_2\ ]/DE$$

and

$$DE = C_1 C_2 + C_c\ (\ C_1 + C_2\ )$$

Applying the waveform relaxation method to Equation (4.10) yields

$$\dot{v}_1^{k+1} = A(1,1)\ v_1^{k+1} + A(1,2)\ v_2^{k}$$

$$\dot{v}_2^{k+1} = A(2,1)\ v_1^{k+1} + A(2,2)\ v_2^{k+1} \tag{4.11}$$

Applying the backward Euler formula to Equation (4.11) yields

$$\begin{bmatrix} 1/h_n - A(1,1) & 0 \\ -A(2,1) & 1/h_n - A(2,2) \end{bmatrix} \begin{bmatrix} v_{1,n+1}^{k+1} \\ v_{2,n+1}^{k+1} \end{bmatrix} = \begin{bmatrix} v_{1,n}^{k+1}/h_n + A(1,2)\, v_{2,n+1}^{k} \\ v_{2,n}^{k+1}/h_n \end{bmatrix} \tag{4.12}$$

Solving Equation (4.12) gives the results in the last column of Table 4.2. It is found that if we process the system from the viewpoint of the state equation instead of the node equation, then the waveform relaxation technique is not affected by the stiffness of the system. Unfortunately, in a practical implementation, it is hard to formulate a large system on the computer in the form of the state equation.

4.3 Concluding Remarks

Table 4.2 shows that the waveform relaxation method does not work well for the analysis of linear stiff systems. The waveforms of each iteration oscillate about the solution, and an excessive number of iterations are required for convergence [21,22], unless the time windows are approximately the size of the smallest time constant of the system. Since the time window is limited to the size of the smallest time constant, in order to achieve convergence in a few iterations, an excessive number of time windows are required to cover one simple transition of the waveform.

The accuracy and convergence properties for the waveform relaxation method have been studied in [9]. In this chapter, we have shown that, in the modified waveform relaxation method, the flexibility of setting the time window saves both the computation time and the memory storage. Although the convergence of the original waveform relaxation method is guaranteed for any arbitrary piecewise continuous set of initial waveforms [9], its convergence is slow in stiff

CHAPTER 5

EVENT-DRIVEN AND LATENCY SCHEMES

Previous works such as MOTIS-C, SPLICE and RELAX showed that considerable improvement in speed can be achieved with the exploitation of relaxation techniques. But they were limited to the simulation of MOS circuits. A dominant factor which yields the performance improvement for these simulators is the use of simplified device models. Hence, in this study we investigate the performance of the time-point relaxation method for the simulation of both MOS and bipolar digital circuit technologies. Performance is also investigated as a function of the device model.

The time-point relaxation method is implemented in the general purpose circuit simulator SLATE [8], and contrary to other relaxation simulators, such as MOTIS, SPLICE, and RELAX, accurate analytical device models are included in the simulator.

5.1 Brief Review of the SLATE Program

The tearing approach [8] decomposes a system into certain subsystems. There are two tearing methods, namely, the node tearing decomposition and the branch tearing decomposition. The former is preferred [8] and is implemented in the SLATE program. As shown in Figure 2.3, the entire circuit is decomposed into several subcir-

tion matrix equation as follows:

$$\begin{bmatrix} Y^{*}_{tt} & Y_{tr} \\ Y_{rt} & Y_{rr} \end{bmatrix} \begin{bmatrix} v_t \\ v_r \end{bmatrix} = \begin{bmatrix} J^{*}_{ts} \\ J_{rs} \end{bmatrix} \tag{5.2}$$

where

$$Y^{*}_{tt} = Y_{tt} - \sum_{i=1}^{k} Y_{tsi}(Y_{si})^{-1}Y_{sti}$$

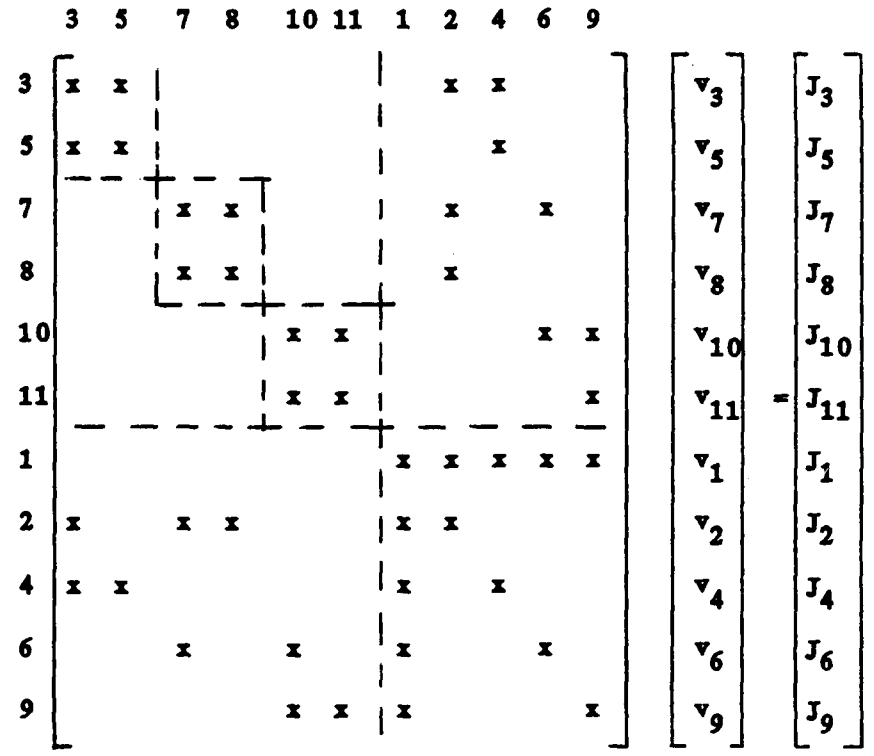
and

$$J^{*}_{ts} = J_{ts} - \sum_{i=1}^{k} Y_{tsi}(Y_{si})^{-1}J_{ssi}$$

Equation (5.2) is solved to obtain solutions for v_t and v_r, and next each subcircuit can be solved by using backward substitution. For example, the resulting modified nodal equation in SLATE for the circuit in Figure 5.1 yields:

systems and requires a large amount of memory, whereas the time-point relaxation technique only suffers the former weakness. Thus, it was concluded that one is better off using the time-point relaxation method in a special purpose circuit simulator for digital circuits.

In the next chapter we describe how the SLATE program [8] was modified to include time-point relaxation. Then its performance for the transient analysis of digital circuits implemented in various technologies is given.



(5.3)

Following the procedure in SLATE, the first step is to LU factor the block triangular terms in order to compute the tearing node voltages v_2, v_4, v_6 and v_9. Then the internal node voltages for each subcircuit can be determined by backward substitution.

In SLATE, because of the use of tearing, only one subcircuit description for each type of repetitive subcircuit need be stored, and only one set of sparse matrix pointers of the small submatrix for each type of repetitive subcircuit is needed so that both storage and preprocessing times can be saved. If one type of subcircuit is linear, then the LU factorization of that type of subcircuit need be

cuits. The nodes of each subcircuit can be classified as the internal nodes and the external (tearing) nodes. After partitioning (which is done by the user), the initial step in the solution strategy of the node tearing method is to reorder the nodes such that all the tearing nodes are located on the border while the internal nodes for each subcircuit are located on the diagonal blocks. The resulting modified nodal equation of the node tearing decomposition from [8] is as follows:

$$\left[\begin{array}{ccccc|c|c} Y_{s1} & & & & & Y_{st1} & \\ & Y_{s2} & & & & Y_{st2} & \\ & & \ddots & & 0 & \vdots & 0 \\ & & & \ddots & & \vdots & \\ & 0 & & & Y_{sk} & Y_{stk} & \\ \hline Y_{ts1} & Y_{ts2} & \cdots & \cdots & Y_{tsk} & Y_{tt} & Y_{tr} \\ & & & 0 & & Y_{rt} & Y_{rr} \end{array}\right] \left[\begin{array}{c} v_{s1} \\ v_{s2} \\ \vdots \\ v_{sk} \\ \hline v_t \\ v_r \end{array}\right] = \left[\begin{array}{c} J_{ss1} \\ J_{ss2} \\ \vdots \\ J_{ssk} \\ \hline J_{ts} \\ J_{rs} \end{array}\right] \qquad (5.1)$$

where v_{si} is the set of the internal nodes for the i-th subcircuit, v_t is the set of all the tearing nodes, and v_r is the set of the nodes for the "rest" of the circuit ("rest": the remaining part of the circuit after all the subcircuits are removed).

The procedures for solving the equations in SLATE are first: eliminate all the Y_{tsi} via LU factorization to get the interconnec-

We use the same circuit in Figure 5.1 to illustrate the partitioning and solution procedures in the implementation of the relaxation technique. In order to apply the modified Gauss-Seidel approach, Equation (5.3) is partitioned as follows:

$$
\begin{array}{c}
\begin{array}{c} \\ 1 \\ 4 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} 1 & 4 & 2 & 3 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\
\left[\begin{array}{ccc:cc:cccc:cc}
x & & x & & & x & & & x & & \\
 & x & & x & x & & & & & & \\
\hdashline
x & & x & x & & & x & x & & & \\
\hdashline
 & x & x & x & x & & & & & & \\
 & x & & x & x & & & & & & \\
\hdashline
x & & & & & x & x & & & x & x \\
 & & x & & & x & x & x & & & \\
 & & x & & & & x & x & & & \\
\hdashline
x & & & & & & & & x & x & \\
 & & & & & x & & & x & x & x \\
 & & & & & x & & & & x &
\end{array}\right]
\end{array}
\left[\begin{array}{c} v_1 \\ v_4 \\ v_2 \\ v_3 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \end{array}\right]
=
\left[\begin{array}{c} J_1 \\ J_4 \\ J_2 \\ J_3 \\ J_5 \\ J_6 \\ J_7 \\ J_8 \\ J_9 \\ J_{10} \\ J_{11} \end{array}\right]
\end{array}
\qquad (5.4)
$$

From the algorithm of the modified Gauss-Seidel method, v_7 and v_8 are relaxed by using a predictor v_7^* for v_7, and a predictor v_8^* for v_8; then, the variables v_2, v_3 and v_5 of the first subcircuit are solved from the 3x3 matrix. The rest of the subcircuits are solved one at a time sequentially in a similar manner. In order to use the same matrix pointers for identical subcircuits, the tearing nodes and the DC power supply nodes are split in program implementation, such

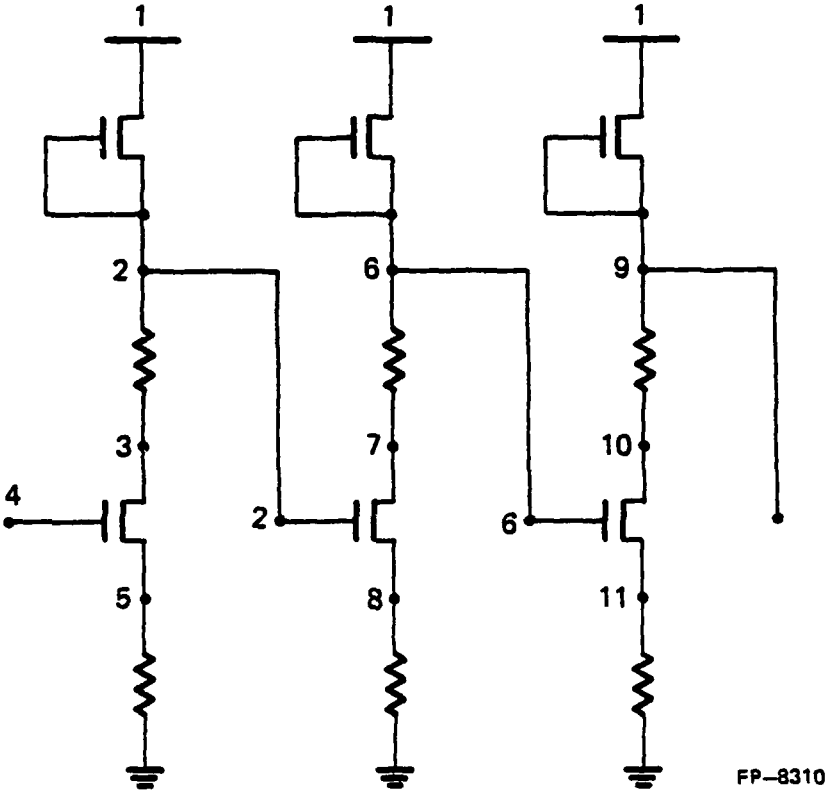


Fig. 5.1 Example Circuit.

clic. From the viewpoint of circuits, the topological ordering is possible only for one-way circuits. If feedback loops exist in the circuit, the directed graph G is no longer acyclic. The directed graph G is said to have strongly connected components (SCC) for circuits with feedback. The strongly connected components of the directed graph G can be found in linear time complexity by using Tarjan's algorithm [24].

For the circuits with feedback loops, basically, there are two approaches for sequencing the vertices in the directed graph G. One is to contract the strongly connected components into one new subcircuit, which results in a new acyclic directed graph G' [25]. The problem with this approach is that the size of the subcircuits after contraction could become too large for the analysis to be efficient. In large-scale circuit simulation one should always try to keep the size of the subcircuits small to make the analysis time linearly proportional to the size of the entire circuit. The other approach processes the directed graph G directly without any contraction by breaking the feedback loops. Predictors are used for the decoupled terms.

The algorithm used in SLATE-R combines the above two approaches. First of all, the Tarjan's algorithm is implemented to detect the strongly connected components (SCC) in the directed graph G and collapses each SCC into one new vertex which results in a new acyclic directed graph G'. Then a topological ordering is chosen as the order in which the vertices in the new directed graph G' are processed dur-

done only once. The other benefit for using the tearing method is the exploitation of latency. During the analysis, only the active parts of the circuit need to be solved and this reduces the computational time considerably.

However, there are additional features that could be added to SLATE to shorten the simulation time. For example, no event scheduling techniques are being used in SLATE, and there is no decoupling scheme in SLATE so that the entire circuit matrix must be inverted. The idea and implementation of the SLATE-R program (a Simulator with Latency and Tearing —Relaxed version) are shown in the next sections.

5.2 The Relaxation Technique

The first procedure of the relaxation decomposition is the same as that of the tearing decomposition: partition the circuit into subcircuits. Current versions of those simulators which use the decomposition technique partition the circuit via the definition of subcircuits that is specified by the user. The only difference in the input processes of the circuit file between SLATE and SLATE-R is SLATE-R has to identify the fan-in and fan-out nodes for each subcircuit while SLATE only needs to record the external nodes no matter if they are fan-in or fan-out nodes. In SLATE-R the circuit input file is changed such that it is easy to identify the fan-in nodes and fan-out nodes. In the next chapter we describe the process of the circuit input file in the SLATE-R program.

We use the same circuit in Figure 5.1 to illustrate the partitioning and solution procedures in the implementation of the relaxation technique. In order to apply the modified Gauss-Seidel approach, Equation (5.3) is partitioned as follows:

$$
\begin{array}{c}
\begin{array}{c} \\ 1 \\ 4 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} 1 & 4 & 2 & 3 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\
\left[\begin{array}{ccccccccccc}
x & & x & & & x & & & x & & \\
 & x & & x & x & & & & & & \\
x & & x & x & & & x & x & & & \\
 & x & x & x & x & & & & & & \\
 & x & & x & x & & & & & & \\
x & & & & & x & x & & & x & x \\
 & & x & & & x & x & x & & & \\
 & & x & & & & x & x & & & \\
x & & & & & & & & x & x & \\
 & & & & & x & & & x & x & x \\
 & & & & & x & & & & x &
\end{array}\right]
\end{array}
\left[\begin{array}{c} v_1 \\ v_4 \\ v_2 \\ v_3 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \end{array}\right]
=
\left[\begin{array}{c} J_1 \\ J_4 \\ J_2 \\ J_3 \\ J_5 \\ J_6 \\ J_7 \\ J_8 \\ J_9 \\ J_{10} \\ J_{11} \end{array}\right]
\end{array}
\qquad (5.4)
$$

From the algorithm of the modified Gauss-Seidel method, v_7 and v_8 are relaxed by using a predictor v_7^* for v_7, and a predictor v_8^* for v_8; then, the variables v_2, v_3 and v_5 of the first subcircuit are solved from the 3x3 matrix. The rest of the subcircuits are solved one at a time sequentially in a similar manner. In order to use the same matrix pointers for identical subcircuits, the tearing nodes and the DC power supply nodes are split in program implementation, such

that each identical subcircuit has the same matrix formation. For the example circuit in Figure 5.1, the subcircuit one contains nodes 1, 2, 3, 4 and 5; subcircuit two contains nodes 1, 6, 7, 2 and 8, with two voltage sources (and two corresponding current variables) in each subcircuit, the relaxation approach needs to solve three 7x7 submatrices, while the SLATE program is to solve one 13x13 matrix.

5.3 Event-Driven Technique

In the solution strategy of the relaxation techniques, once the whole circuit has been partitioned into several subcircuits, the next procedure, which is called the event-driven technique, is to sequence the subcircuits to be simulated, i.e., to select a well-chosen ordering in which the subcircuits are to be processed. In our research, event-driven algorithms will be implemented on the basis of fan-in fan-out topologies, such that the resulting modified nodal equation is in the form of Equation (5.4) with the internal nodes and fan-out nodes of each subcircuit in diagonal blocks.

A circuit that is composed of unilateral subcircuits can be represented by a directed graph G(V,E), where each vertex in V corresponds to each subcircuit and each edge in E corresponds to each signal line from fan-out to fan-in. A "good" ordering will be one in which a subcircuit is processed only after all its fan-in subcircuits have already been processed; in other words, the event-driven technique is to arrange the vertices in a topological order. However, the directed graph G has a topological ordering if and only if it is acy-

clic. From the viewpoint of circuits, the topological ordering is possible only for one-way circuits. If feedback loops exist in the circuit, the directed graph G is no longer acyclic. The directed graph G is said to have strongly connected components (SCC) for circuits with feedback. The strongly connected components of the directed graph G can be found in linear time complexity by using Tarjan's algorithm [24].

For the circuits with feedback loops, basically, there are two approaches for sequencing the vertices in the directed graph G. One is to contract the strongly connected components into one new subcircuit, which results in a new acyclic directed graph G' [25]. The problem with this approach is that the size of the subcircuits after contraction could become too large for the analysis to be efficient. In large-scale circuit simulation one should always try to keep the size of the subcircuits small to make the analysis time linearly proportional to the size of the entire circuit. The other approach processes the directed graph G directly without any contraction by breaking the feedback loops. Predictors are used for the decoupled terms.

The algorithm used in SLATE-R combines the above two approaches. First of all, the Tarjan's algorithm is implemented to detect the strongly connected components (SCC) in the directed graph G and collapses each SCC into one new vertex which results in a new acyclic directed graph G'. Then a topological ordering is chosen as the order in which the vertices in the new directed graph G' are processed dur-

ing each iteration. In order to set up the order in which the subcircuits within each SCC are processed, the following algorithm [26] is implemented to set up the analysis sequences. In order to describe the algorithm, the following notations are introduced:

$G_{scc}(V,E)$: the directed graph of each SCC.

adj(v): set of adjacent vertices corresponds to the set of vertices with an edge which fans out to vertex v.

la(v): label of vertex v.

Procedure

[1] Set $la(v_i) = 0$ for each vertex v_i of $G_{scc}(V,E)$

[2] 2. $la(v_i)=1$ for each vertex v_i which corresponds to an input signal terminal.

k=1.

[3] k=k+1

Choose a vertex v_j where $la(v_j)=0$ and $la(v_i)\neq 0$ for all v_i ϵ $adj(v_j)$. If there is no such vertex, choose a vertex v_j connecting to a vertex which has the lowest label.

$la(v_j)=k$.

[4] Repeat step (3) until all the vertices in $G_{scc}(V,E)$ are labeled.

It is claimed in [26] that the above algorithm can find all feedback loops which will be broken during the analysis sequencing.

For arbitrary networks, this algorithm may not be satisfactory in identifying minimal feedback loops as other complex algorithms do [27]. However, because the accurate device models are used in SLATE, coupling parameters such as the floating capacitor from drain to gate in MOS devices and the terminal resistance at each leg of the bipolar transistor devices will cause the subcircuits to have a certain degree of coupling. In this case, our solution strategy is to label the vertices in sense of "depth-first search" approach and to use a predictor to cut all the feedback loops. Another fact is that iterations among subcircuits are continued until convergence is reached. The worst case for "improper" ordering is at most the necessity to process one more iteration at each time point. General algorithms are not cost effective because the complexity grows exponentially with the size of the network [27]. For these two reasons we implement the simple analysis sequencing algorithm in our program. In the next section, we will describe the concept and experimental results of latency exploitation.

5.4 Latency Exploitation

In large-scale circuit analysis, a part of the circuit may be not active at any given time and at any particular iteration. This phenomenon is called latency [28]. Exploitation of the latency can provide savings in CPU time. In the program SLATE, the latency concept is implemented at three levels: (1) device level; (2) subcircuit level; and (3) network level.

At the device level, the operating point of each nonlinear device is monitored. If the operating point does not change significantly between time points or Newton-Raphson iterations, the device models need not be reevaluated, and the matrix entries computed at the previous time point or the previous iteration are used again. This level of latency is also called the device bypass level which is used in SPICE2 and SPLICE. Because the node tearing method is implemented in SLATE, the latency strategy can be used in the network level through the exploitation of the substitution theorem in the formulation of interconnections [8]. While with the relaxation method, there is no bordered interconnection matrix, in the SLATE-R program we only consider the exploitation of latency at the bypass level and subcircuit level.

The relaxation method deals with each subcircuit individually, with the use of the predictor to decouple the coupling terms. Therefore the exploitation of the latency can be implemented at the subcircuit level, either in the Newton-Raphson iteration or at the time point level. The idea is that if there is no significant change between Newton-Raphson iterations or time points, then the latent subcircuit can be skipped in the analysis. The latent status of a subcircuit can be checked by monitoring the changes of all its stimuli and all its responses to ensure that the change is within certain predetermined tolerances.

5.4.1 The Latency Criterion

For the subcircuit N_k, denote the fan-in node voltages as v_{ik_p}, p=1,2,...., and the internal and output node voltages of N_k as v_{ok_q}, q=1,2,.. . The following latency scheme is implemented in the SLATE-R program.

Latency Scheme:

A subcircuit N_k is considered to be latent at time t_n if the following two conditions are satisfied:

[1] $$| v_{ik_p}(t_n) - v_{ik_p}(t_{n-1}) | < \varepsilon_a + \varepsilon_r \max(|v_{ik_p}(t_n)|, |v_{ik_p}(t_{n-1})|$$

p=1,2,...

[2] $$| v_{ok_q}(t_n) - v_{ok_q}(t_{n-1}) | < \varepsilon_a + \varepsilon_r \max(|v_{ok_q}(t_n)|, |v_{ok_q}(t_{n-1})|$$

q=1,2,...

The subcircuit N_k will remain latent at time t_{n+j} as long as

$$| v_{ik_p}(t_{n+j}) - v_{ik_p}(t_{n-1}) | < \varepsilon_a +$$

$$\varepsilon_r \max (| v_{ik_p}(t_{n+j}) |, | v_{ik_p}(t_{n-1}) |$$

p=1,2,...

If the subcircuit is not latent in time, we can still consider the latency in Newton-Raphson iterations. A subcircuit N_k is considered to be latent at time t_n during the ith Newton-Raphson

iterations if the following two conditions are satisfied:

$$[1] \quad | v_{ik_p}(t_n, i-1) - v_{ik_p}, i-2) | < \varepsilon_a + \varepsilon_r \max (| v_{ik_p}(t_n, i-1) |, | v_{ik_p}(t_n, i-2) |$$

$$p=1,2,\ldots$$

$$[2] \quad | v_{ok_q}(t_n, i-1) - v_{ok_q}, i-2) | < \varepsilon_a + \varepsilon_r \max (| v_{ok_q}(t_n, i-1) |, | v_{ok_q}(t_n, i-2) |$$

$$q=1,2,\ldots$$

5.4.2 Experimental Results for Latency Exploitation

The latency scheme described in the last section has been successfully implemented into the relaxation version of the program SLATE. Table 5.1 and Table 5.2 give the simulation data corresponding to the circuits shown in Figure 5.2 and Figure 5.3 respectively. In order to see the latency exploitation at the Newton-Raphson iteration level, here the dc analysis is performed, while Table 5.3 gives the simulation data for the circuit shown in Figure 5.4 for both DC analysis and transient analysis.

Table 5.1 Simulation data of a DC analysis
for the MOS circuit in Figure 5.2

DC Analysis	With Latency	Without Latency
# of subcircuits times # of iterations	275	275
# of nonlatent subcircuits times # of iterations	157	
Latency exploitation (%)	42.91	
Total CPU time (sec)	5.63	7.68
Savings in CPU time (%)	26.69	

Table 5.2 Simulation data of a DC analysis
for the MOS circuit in Figure 5.3

DC Analysis	With Latency	Without Latency
# of subcircuits times # of iterations	391	391
# of nonlatent subcircuits times # of iterations	214	
Latency exploitation (%)	45.27	
Total CPU time (sec)	11.87	14.82
Savings in CPU time (%)	19.91	

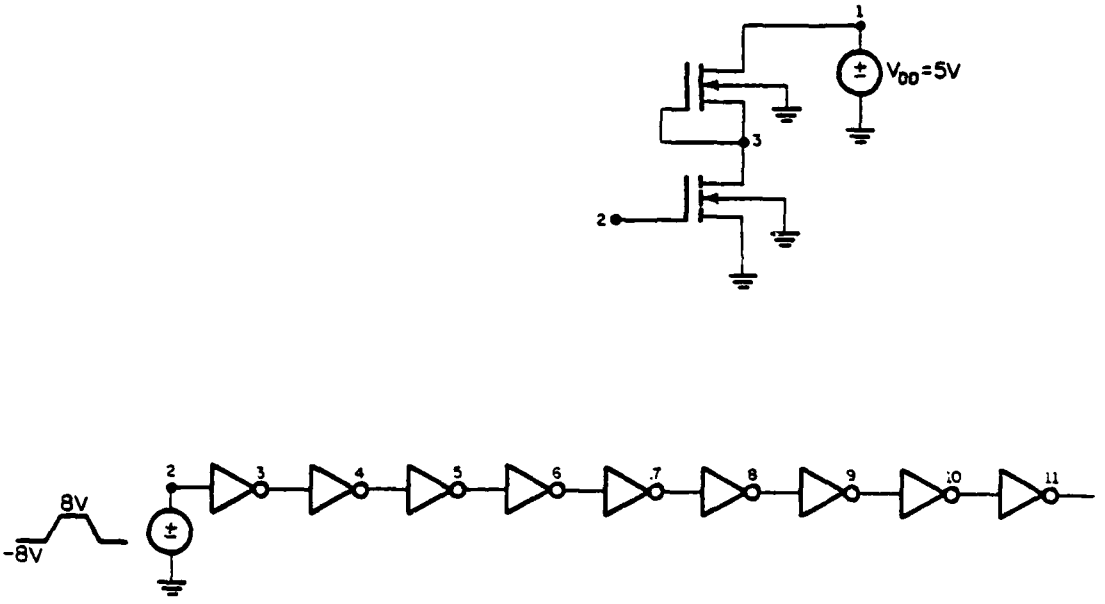
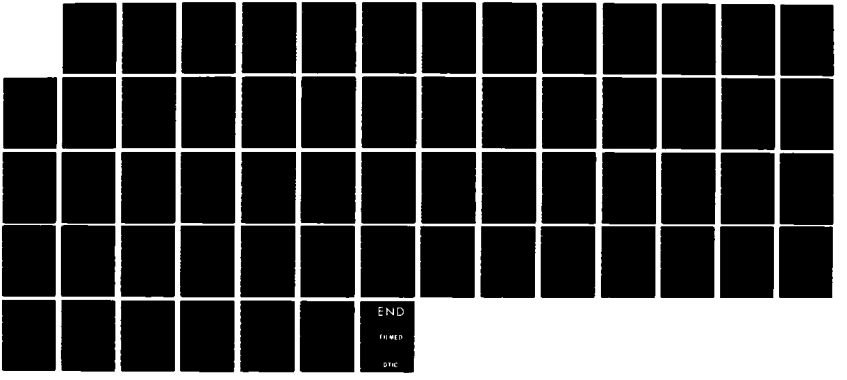
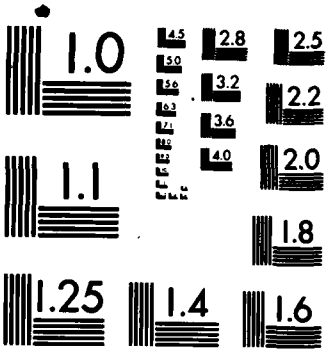


Fig. 5.2 (a) Subcircuit: An MOS Inverter Gate.
(b) Entire Network: A Chain of Inverters.

AD-A161 854 A STUDY OF RELAXATION TECHNIQUES FOR THE TRANSIENT ANALYSIS OF DIGITAL CI (U) ILLINOIS UNIV CHAMPAIGN COGNITIVE PSYCHOPHYSIOLOGY LAB W CHIA 03 JUN 85 2/2

UNCLASSIFIED UILU-ENG-85-2203 N00014-84-C-0149 F/G 9/3 NL





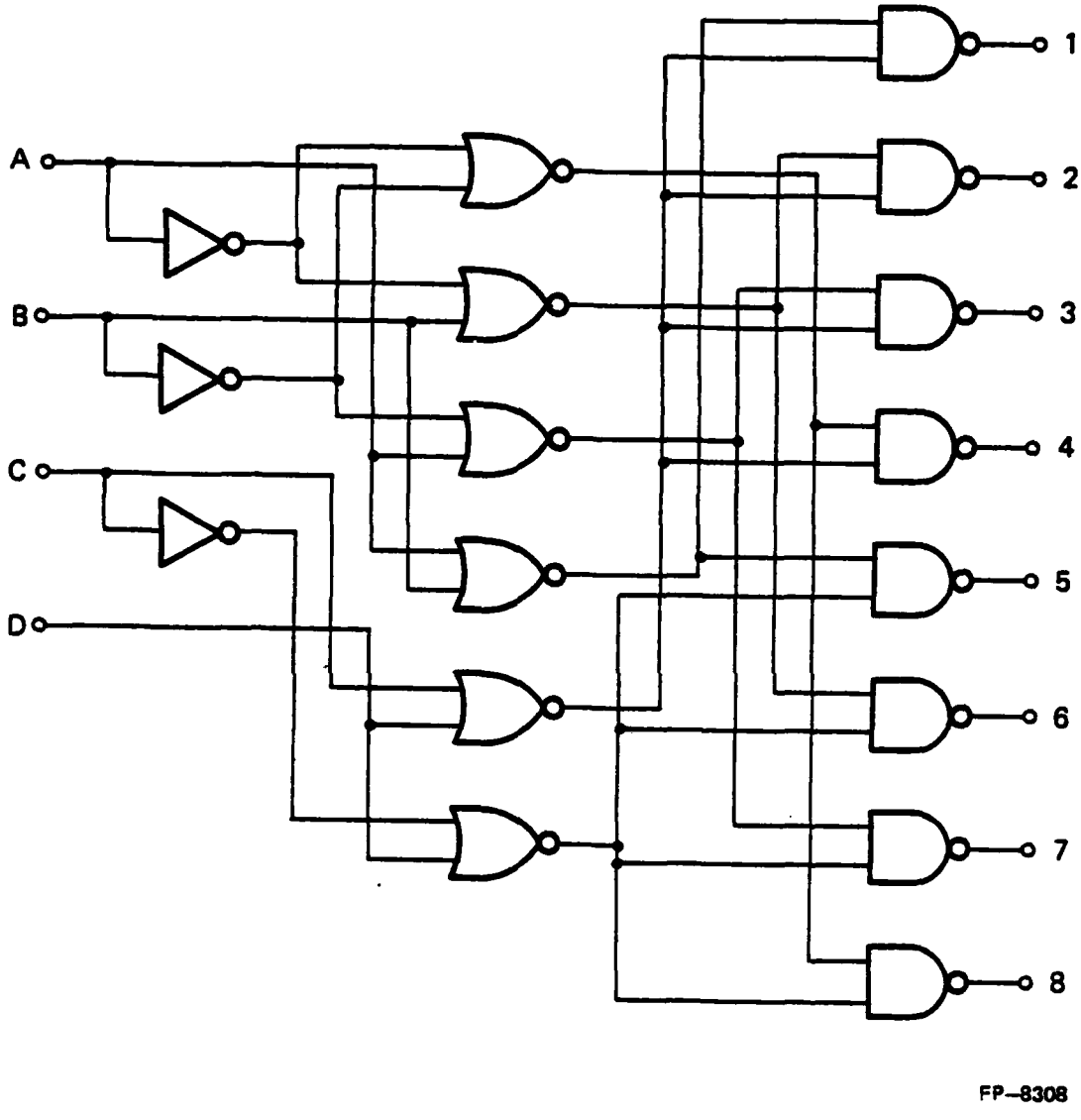


Fig. 5.3 Block Diagram of Binary-to-Octal Decoder.

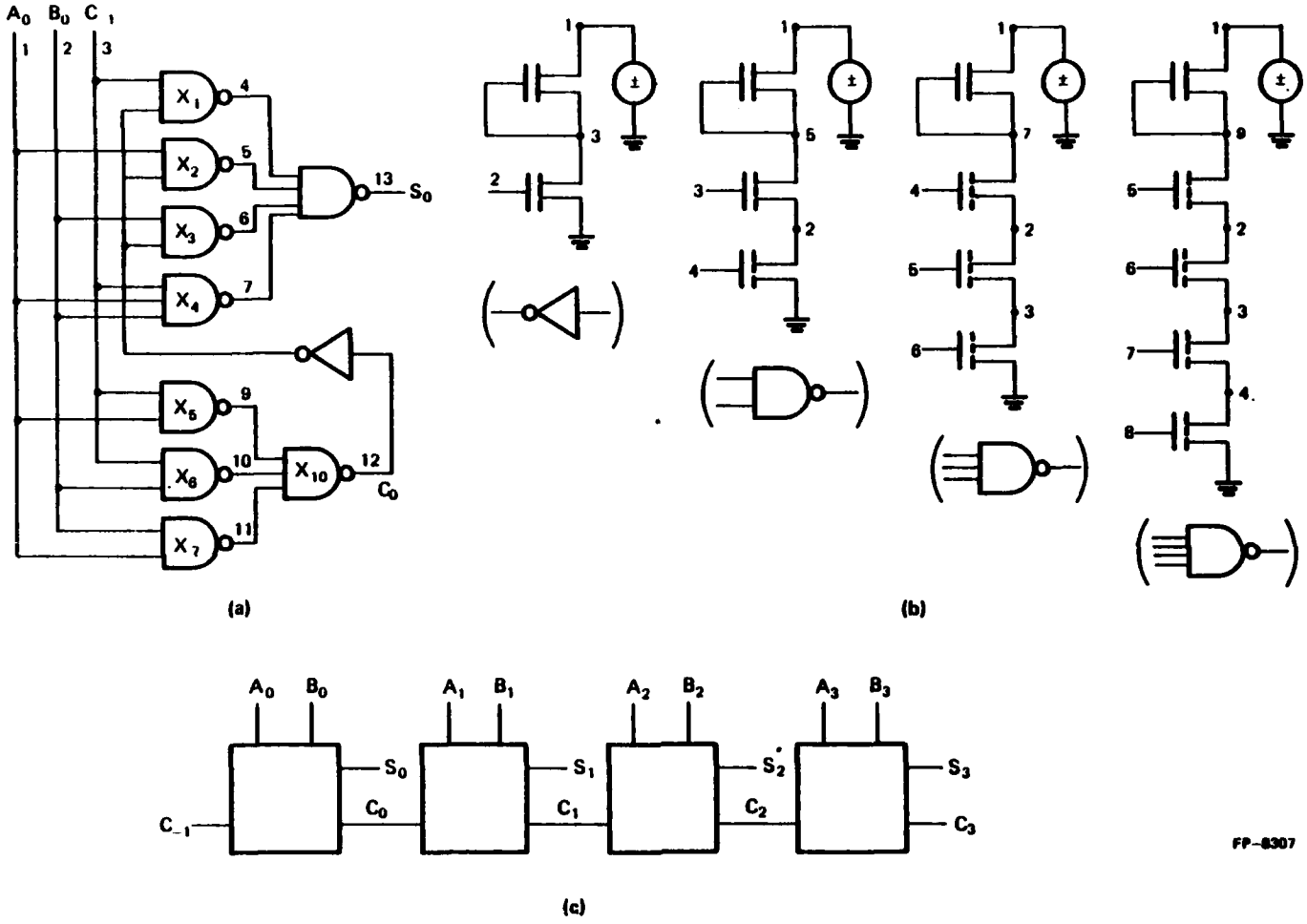


Fig. 5.4 (a) Subcircuits. (b) One Bit Full Adder. (c) Four Bit Adder.

Remark:

In these tables we find that the savings in CPU time is not the same for the latency exploitation. For example, in Table 5.1, for the 11-stage chain of inverters, a 42.91% latency exploitation was achieved and a 26.69% savings in CPU time was obtained, because some of the CPU time has to be spent in the latency checking which makes the difference between the percentage latency exploitation and the percentage savings in the CPU time. In Table 5.2, for the Binary-to-Octal decoder, a 45.27% latency exploitation was achieved and a 19.91% savings in CPU time was obtained, because the latency exploitation only counts the number of those latent subcircuits, and all the subcircuits are preassumed to have the same size. However, the subcircuits may have different sizes which also make the difference between the latency exploitation and the CPU time savings. From Table 5.3, for the one-bit full adder, a 14.75% savings in CPU time was achieved in DC analysis and a 22.91% savings in transient analysis was achieved, because in transient analysis we can take the advantages of the latency exploitation in the Newton-Raphson iteration level as well as the advantages of latency exploitation in the time level.

Table 5.3a Simulation data of a DC analysis for the MOS circuit in Figure 5.4

DC Analysis	With Latency	Without Latency
# of subcircuits times # of iterations	150	150
# of nonlatent subcircuits times # of iterations	107	
Latency exploitation (%)	28.67	
Total CPU time (sec)	6.18	7.25
Savings in CPU time (%)	14.75	

Table 5.3b Simulation data of a transient analysis for the MOS circuit in Figure 5.4

Transient Analysis	With Latency	Without Latency
# of subcircuits times # of iterations	840	840
# of nonlatent subcircuits times # of iterations	597	
Latency exploitation (%)	28.92	
Total CPU time (sec)	31.95	41.45
Savings in CPU time (%)	22.91	

5.5 Numerical Properties

In Chapter 3, the linear test circuit in Figure 3.1 has been used to study the numerical properties of the Gauss-Seidel techniques. This test circuit was generated by linearizing the model of a cascade of two inverters in which the transistors are assumed to be active. By changing the parameters, i.e., the coupling term C_c and the signal gain g_m, we can adjust the degree of stiffness in our linear test circuit.

In practical circuits, the coupling terms and the signal gains are technology-oriented. In this section, different technologies such as NMOS, CMOS and bipolar junction transistors are used to implement the cascade of a two inverter test circuit. The NMOS inverters and CMOS inverters are shown in Figure 5.5 and Figure 5.6, respectively. Figure 5.7 shows the TTL circuit, Figure 5.8. shows the ECL circuit. Table 5.4, Table 5.5, Table 5.6 and Table 5.7 show the corresponding simulation data.

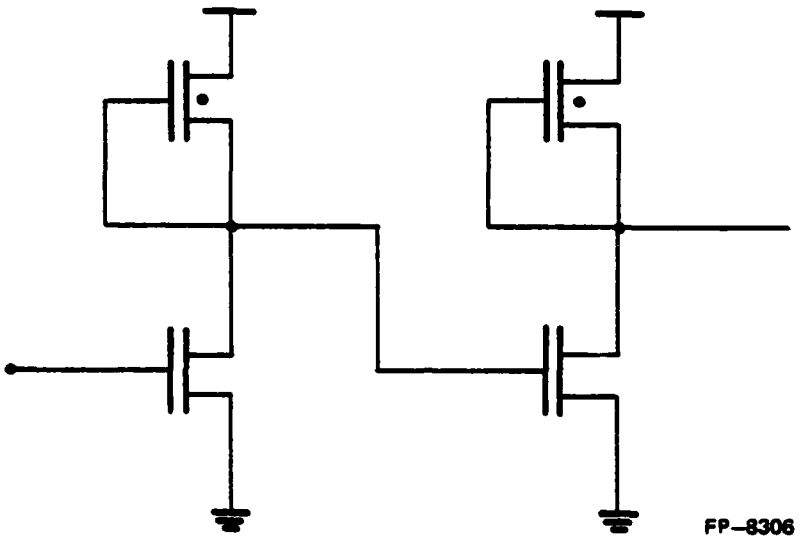


Fig. 5.5 NMOS Circuits.

The input file for the circuit in Fig. 5.5

```
a 11-stage chain inverter circuit
*inverter
.subckt inv 10 30 20 2
*** nodes: vdd input output
m1 10 20 20 0  dm w=5 l=10
+              as=25 ad=25 ass=15 asd=20 cgs=1.725f cgd=1.725f
+              rdd=35 rss=35
m2 20 30 0  0  em w=10 l=5
+              as=100 ad=100 ass=40 asd=35 cgs=3.45f cgd=3.45f
+              rdd=35 rss=35
.ends
* nominal circuit
vdd 10 0 5
vc1 9 0 pulse(0 5 0 2n 2n 125n 254n)
x1  10  9 11            inv
x2  10 11 12            inv
.model dm nmos vto=-2 kp=10u be=0.52 lambda=0.05 kpn=0.0918f
+              ns=0.33 cox=0.345f
.model em nmos vto=1 kp=10u be=0.52 lambda=0.05 kpn=0.0918f
+              ns=0.33 cox=0.345f
.print tran v(9) v(10) v(11) v(12)
.tran 1n,50n
.end
```

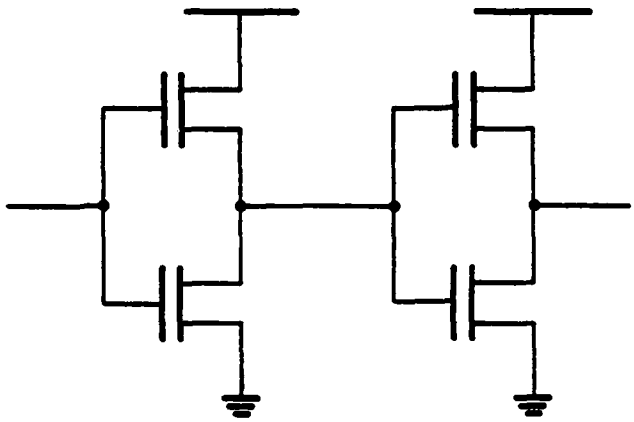


Fig. 5.6 CMOS Circuits.

The input file for the circuit in Fig. 5.6

```
 a 11-stage chain inverter circuit
 *inverter
.subckt inv 10 20 30
 *** nodes: vdd input output
m1 10 20 30 10 cm1 w=5 l=5
+               as=25 ad=25 ass=15 asd=20 cgs=1.725f cgd=1.725f
+               rdd=35 rss=35
m2 30 20 0  0  cm2 w=12.5 l=5
+               as=100 ad=100 ass=40 asd=35 cgs=3.45f cgd=3.45f
+               rdd=35 rss=35
.ends
 * nominal circuit
 vdd 10 0 5
 vc1 9 0 pulse(5 0 0 30n 2n 60n 40n)
 x1  10  9 11            inv
 x2  10 11 12            inv
.model cm1 pmos vto=-1 kp=8u be=0.52 lambda=0.05 kpn=0.0918f
+               ns=0.33 cox=0.345f
.model cm2 nmos vto=1 kp=20u be=0.52 lambda=0.05 kpn=0.0918f
+               ns=0.33 cox=0.345f
.print tran v(9) v(10) v(11) v(12)
.tran 1n,50n
.end
```

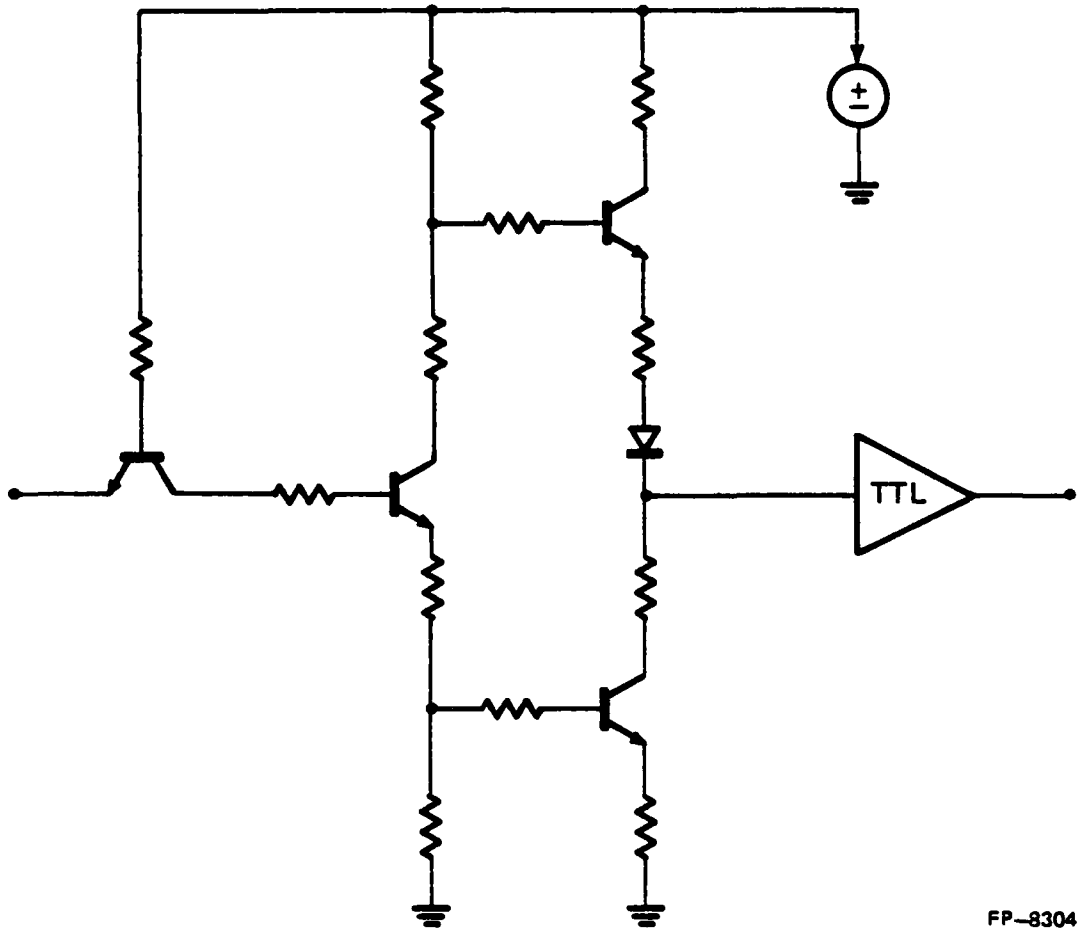


Fig. 5.7 TTL Inverters.

The input file for the circuit in Fig. 5.7

```
Bipolar TTL inverter
.subckt inv 1 9 8 2
* input node(1) output node(8)
r1 9 2 4k
r2 4 0 1k
r3 9 5 1.4k
r4 9 6 100
q1 3 2 1 bjp
q2 5 3 4 bjp
q3 6 5 7 bjp
q4 8 4 0 bjp
d1 7 8 diode
.ends
x1 2 9 3 inv
x2 3 9 4 inv
v2 2 0 pulse(0 5 0 30n 2n 60n 40n)
vdd 9 0 5
.model bjp npn(bf=100 br=0.1 re=100 va=200 nc=1 ne=1
+              cje=2p cjc=2p ccs=2p)
.model diode d(rs=10)
.print tran v(2) v(3) v(4)
.tran 1n,30n
.end
```

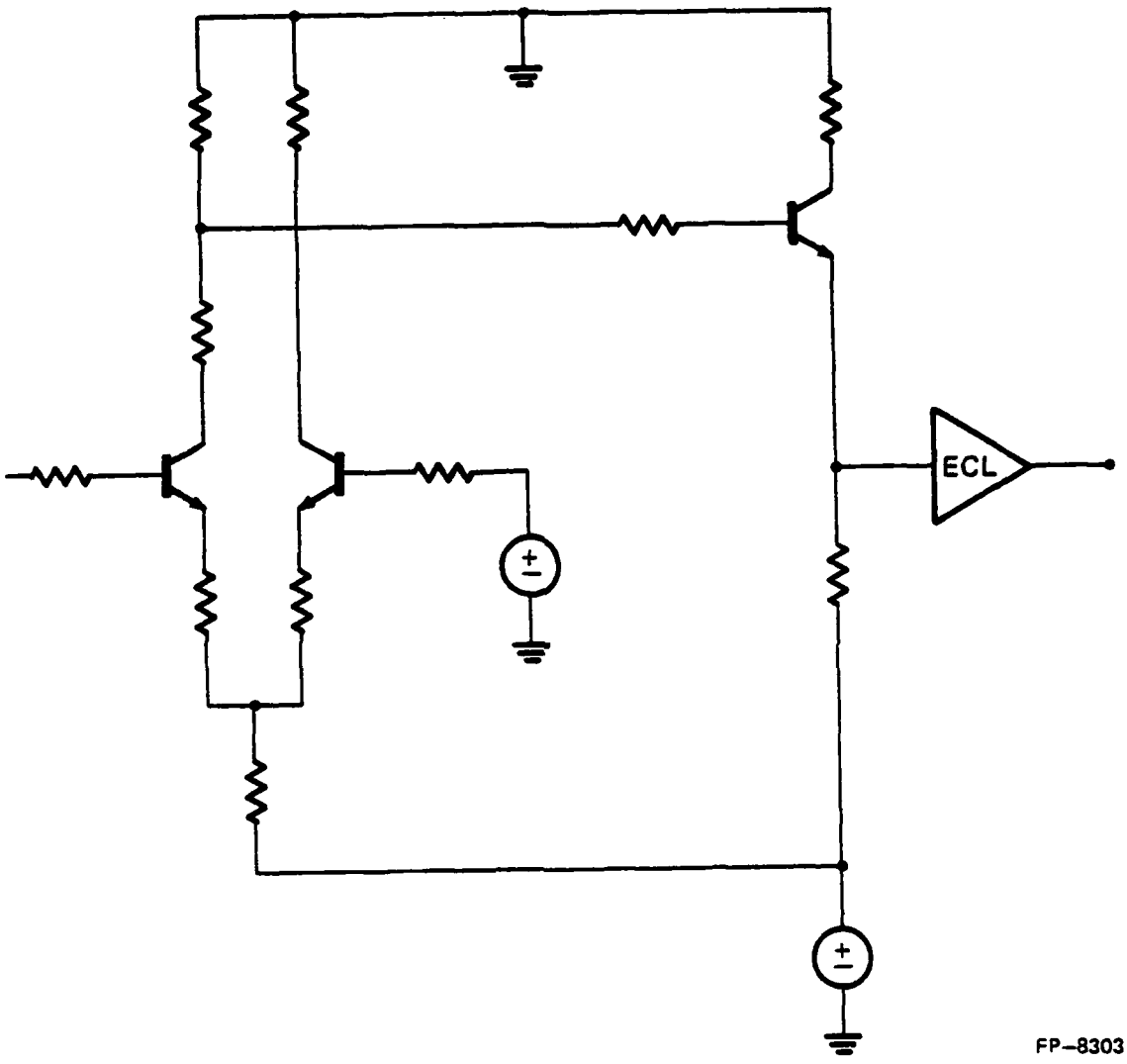


Fig. 5.8 ECL Inverters.

The input file for the circuit in Fig. 5.8

```
Bipolar ECL inverter
.subckt inv 7 8
* input node(7) output node(8)
r1 1 0 300
r2 2 0 300
r3 3 5 1.18k
r4 5 8 1.5k
q2 1 7 3 bjp
q3 2 4 3 bjp
q4 0 1 8 bjp
vrr 4 0 -1.1
vdd 5 0 -5.2
.ends
x1 2 3 inv
x2 3 4 inv
v2 2 0 exp(-1.5 1.5  0 30n 2n 60n 40n)
.model bjp npn(bf=100 br=0.1 rb=200 va=200 nc=1 ne=1
+              cje=2p cjc=2p ccs=2p)
.print tran v(2) v(3) v(4)
.tran 1n,30n
.end
```

Table 5.4a

NMOS inverters with signal gain = 9.5

(SLATE-R)

	time	time step	coupling	# of iterations
0	0	0	0	11
1	.5000e-10	.5000e-10	.48298e-03	3
2	.1000e-09	.5000e-10	.48298e-03	3
3	.3000e-09	.2000e-09	.24149e-03	3
4	.1100e-08	.8000e-09	.60374e-04	4
5	.2000e-08	.9000e-09	.53664e-04	5
6	.2050e-08	.5000e-10	.48297e-03	3
7	.2100e-08	.5000e-10	.96593e-03	3
8	.2300e-08	.2000e-09	.24147e-03	3
9	.3100e-08	.8000e-09	.60373e-04	4
10	.5100e-08	.2000e-08	.23489e-04	6
11	.7100e-08	.2000e-08	.19379e-04	4
12	.9100e-08	.2000e-08	.18400e-04	4
13	.1110e-08	.2000e-08	.18400e-04	4
14	.1310e-08	.2000e-08	.18400e-04	4
15	.1510e-08	.2000e-08	.18400e-04	4
16	.1710e-08	.2000e-08	.18400e-04	4
17	.1910e-08	.2000e-08	.18400e-04	4
18	.2110e-07	.2000e-08	.18400e-04	4
19	.2310e-07	.2000e-08	.18400e-04	4
20	.2510e-07	.2000e-08	.18400e-04	3
21	.2710e-07	.2000e-08	.18400e-04	3
22	.2910e-07	.2000e-08	.18400e-04	3
23	.3110e-07	.2000e-08	.18400e-04	3
24	.3310e-07	.2000e-08	.18400e-04	3
25	.3510e-07	.2000e-08	.18400e-04	3
26	.4710e-07	.2000e-08	.18400e-04	3
27	.3910e-07	.2000e-08	.18400e-04	3
28	.4110e-07	.2000e-08	.18400e-04	3
29	.4310e-07	.2000e-08	.18400e-04	3
30	.4510e-07	.2000e-08	.18400e-04	3
31	.4710e-07	.2000e-08	.18400e-04	3
32	.4910e-07	.2000e-08	.18400e-04	3
33	.5000e-07	.9000e-09	.40889e-04	3

Table 5.4b

NMOS inverters with signal gain = 9.5

(SLATE)

	time	time step	# of iterations
0	0	0	11
1	.5000e-10	.5000e-10	3
2	.1000e-09	.5000e-10	2
3	.3000e-09	.2000e-09	2
4	.1100e-08	.8000e-09	4
5	.2000e-08	.9000e-09	5
6	.2050e-08	.5000e-10	3
7	.2100e-08	.5000e-10	2
8	.2300e-08	.2000e-09	3
9	.3100e-08	.8000e-09	3
10	.5100e-08	.2000e-08	4
11	.7100e-08	.2000e-08	3
12	.9100e-08	.2000e-08	3
13	.1110e-08	.2000e-08	3
14	.1310e-08	.2000e-08	3
15	.1510e-08	.2000e-08	3
16	.1710e-08	.2000e-08	3
17	.1910e-08	.2000e-08	3
18	.2110e-07	.2000e-08	3
19	.2310e-07	.2000e-08	3
20	.2510e-07	.2000e-08	3
21	.2710e-07	.2000e-08	3
22	.2910e-07	.2000e-08	2
23	.3110e-07	.2000e-08	2
24	.3310e-07	.2000e-08	2
25	.3510e-07	.2000e-08	2
26	.4710e-07	.2000e-08	2
27	.3910e-07	.2000e-08	2
28	.4110e-07	.2000e-08	2
29	.4310e-07	.2000e-08	2
30	.4510e-07	.2000e-08	2
31	.4710e-07	.2000e-08	2
32	.4910e-07	.2000e-08	2
33	.5000e-07	.9000e-09	2

Table 5.5a

CMOS inverters with signal gain = 11.7

(SLATE-R)

	time	time step	coupling	# of iterations
0	0	0	0	13
1	.5000e-10	.5000e-10	.81705e-03	5
2	.1000e-09	.5000e-10	.81705e-03	4
3	.3000e-09	.2000e-09	.40537e-03	4
4	.1100e-08	.8000e-09	.83374e-04	5
5	.3100e-08	.2000e-08	.33350e-04	4
6	.5100e-08	.2000e-08	.33350e-04	4
7	.7100e-08	.2000e-08	.33350e-04	4
8	.9100e-08	.2000e-08	.33350e-04	4
9	.1110e-07	.2000e-08	.33350e-04	4
10	.1310e-07	.2000e-08	.33350e-04	4
11	.1510e-07	.2000e-08	.33350e-04	4
12	.1710e-07	.2000e-08	.33333e-04	7
13	.1910e-07	.2000e-08	.37545e-04	7
14	.2110e-07	.2000e-08	.37662e-04	5
15	.2310e-07	.2000e-08	.37662e-04	3
16	.2510e-07	.2000e-08	.37662e-04	5
17	.2710e-07	.2000e-08	.37662e-04	3
18	.2910e-07	.2000e-08	.37662e-04	4
19	.3000e-07	.9000e-09	.83674e-04	3
20	.3050e-07	.5000e-10	.75325e-03	4
21	.3010e-07	.5000e-10	.15065e-02	4
22	.3030e-07	.2000e-09	.37662e-03	4
23	.3110e-07	.8000e-09	.94155e-04	4
24	.3310e-07	.2000e-08	.37662e-04	4
25	.3510e-07	.2000e-08	.37662e-04	3
26	.4710e-07	.2000e-08	.37662e-04	3
27	.3910e-07	.2000e-08	.37662e-04	3
28	.4110e-07	.2000e-08	.37662e-04	3
29	.4310e-07	.2000e-08	.37662e-04	3
30	.4510e-07	.2000e-08	.37662e-04	3
31	.4710e-07	.2000e-08	.37662e-04	3
32	.4910e-07	.2000e-08	.37662e-04	2
33	.5000e-07	.9000e-09	.83694e-04	3

Table 5.5b

CMOS inverters with signal gain = 11.7

(SLATE)

	time	time step	# of iterations
0	0	0	13
1	.5000e-10	.5000e-10	2
2	.1000e-09	.5000e-10	2
3	.3000e-09	.2000e-09	3
4	.1100e-08	.8000e-09	3
5	.3100e-08	.2000e-08	3
6	.5100e-08	.2000e-08	3
7	.7100e-08	.2000e-08	3
8	.9100e-08	.2000e-08	3
9	.1110e-07	.2000e-08	3
10	.1310e-07	.2000e-08	3
11	.1510e-07	.2000e-08	3
12	.1710e-07	.2000e-08	4
13	.1910e-07	.2000e-08	5
14	.2110e-07	.2000e-08	5
15	.2310e-07	.2000e-08	3
16	.2510e-07	.2000e-08	7
17	.2710e-07	.2000e-08	3
18	.2910e-07	.2000e-08	3
19	.3000e-07	.9000e-09	3
20	.3050e-07	.5000e-10	3
21	.3010e-07	.5000e-10	2
22	.3030e-07	.2000e-09	3
23	.3110e-07	.8000e-09	3
24	.3310e-07	.2000e-08	3
25	.3510e-07	.2000e-08	3
26	.4710e-07	.2000e-08	3
27	.3910e-07	.2000e-08	2
28	.4110e-07	.2000e-08	2
29	.4310e-07	.2000e-08	2
30	.4510e-07	.2000e-08	2
31	.4710e-07	.2000e-08	2
32	.4910e-07	.2000e-08	2
33	.5000e-07	.9000e-09	2

Table 5.6a

TTL inverters with re=100

(SLATE-R)

	time	time step	coupling	# of iterations
0	0	0	.10000e-01	68
1	.5000e-10	.5000e-10	.10000e-01	4
2	.1000e-09	.5000e-10	.10000e-01	4
3	.3000e-09	.2000e-09	.10000e-01	4
4	.1100e-08	.8000e-09	.10000e-01	5
5	.3100e-08	.2000e-08	.10000e-01	8
6	.5100e-08	.2000e-08	.10000e-01	5
7	.7100e-08	.2000e-08	.10000e-01	6
8	.9100e-08	.2000e-08	.10000e-01	7
9	.1110e-07	.2000e-08	.10000e-01	8
10	.1310e-07	.2000e-08	.10000e-01	9
11	.1510e-07	.2000e-08	.10000e-01	11
12	.1710e-07	.2000e-08	.10000e-01	10
13	.1910e-07	.2000e-08	.10000e-01	8
14	.2110e-07	.2000e-08	.10000e-01	8
15	.2310e-07	.2000e-08	.10000e-01	8
16	.2510e-07	.2000e-08	.10000e-01	9
17	.2710e-07	.2000e-08	.10000e-01	7
18	.2910e-07	.2000e-08	.10000e-01	9
19	.3000e-07	.9000e-09	.10000e-01	7

The coupling is the conductance of the emitter resistance re=100.

Table 5.6b

TTL inverters with re=100

(SLATE)

	time	time step	# of iterations
0	0	0	
1	.5000e-10	.5000e-10	4
2	.1000e-09	.5000e-10	3
3	.3000e-09	.2000e-09	4
4	.1100e-08	.8000e-09	5
5	.3100e-08	.2000e-08	8
6	.5100e-08	.2000e-08	5
7	.7100e-08	.2000e-08	7
8	.9100e-08	.2000e-08	5
9	.1110e-07	.2000e-08	13
10	.1310e-07	.2000e-08	6
11	.1510e-07	.2000e-08	11
12	.1710e-07	.2000e-08	12
13	.1910e-07	.2000e-08	6
14	.2110e-07	.2000e-08	6
15	.2310e-07	.2000e-08	7
16	.2510e-07	.2000e-08	9
17	.2710e-07	.2000e-08	7
18	.2910e-07	.2000e-08	8
19	.3000e-07	.9000e-09	5

Table 5.6c

TTL inverters with re=10

(SLATE-R)

	time	time step	coupling	# of iterations	
0	0	0	.10000e-01	40	
1	.5000e-10	.5000e-10	.10000e-00	5	
2	.1000e-09	.5000e-10	.10000e-00	5	
3	.3000e-09	.2000e-09	.10000e-00	5	
4	.1100e-08	.8000e-09	.10000e-00	7	
5	.3100e-08	.2000e-08	.10000e-00	7	
6	.5100e-08	.2000e-08	.10000e-00	7	
7	.7100e-08	.2000e-08	.10000e-00	7	
8	.7350e-08	.2500e-09	.10000e-00	7	(*)
9	.8350e-07	.1000e-08	.10000e-00	29	
10	.1035e-07	.2000e-08	.10000e-00	18	
11	.1235e-07	.2000e-08	.10000e-00	13	
12	.1435e-07	.2000e-08	.10000e-00	14	
13	.1635e-07	.2000e-08	.10000e-00	9	
14	.1835e-07	.2000e-08	.10000e-00	9	
15	.2035e-07	.2000e-08	.10000e-00	7	
16	.2235e-07	.2000e-08	.10000e-00	7	
17	.2435e-07	.2000e-08	.10000e-00	9	
18	.2635e-07	.2000e-08	.10000e-00	7	
19	.2835e-07	.2000e-08	.10000e-00	9	
20	.3000e-07	.1650e-09	.10000e-00	9	

(*): The original time step can not converge within 50 iterations, with the reduced time step being used.

The coupling is the conductance of the emitter resistance re=10.

Table 5.6d

TTL inverters with re=10

(SLATE)

	time	time step	# of iterations
0	0	0	35
1	.5000e-10	.5000e-10	4
2	.1000e-09	.5000e-10	4
3	.3000e-09	.2000e-09	4
4	.1100e-08	.8000e-09	7
5	.3100e-08	.2000e-08	7
6	.5100e-08	.2000e-08	5
7	.7100e-08	.2000e-08	7
8	.9100e-08	.2000e-08	12
9	.1110e-07	.2000e-08	7
10	.1310e-07	.2000e-08	13
11	.1510e-07	.2000e-08	11
12	.1710e-07	.2000e-08	10
13	.1910e-07	.2000e-08	15
14	.2110e-07	.2000e-08	6
15	.2310e-07	.2000e-08	8
16	.2510e-07	.2000e-08	6
17	.2710e-07	.2000e-08	6
18	.2910e-07	.2000e-08	6
19	.3000e-07	.9000e-09	6

Table 5.6e

TTL inverters with re=1

(SLATE-R)

	time	time step	coupling	# of iterations	
0	0	0	.10000e-01	65	
1	.5000e-10	.5000e-10	.10000e+01	5	
2	.1000e-09	.5000e-10	.10000e+01	6	
3	.3000e-09	.2000e-09	.10000e+01	6	
4	.1100e-08	.8000e-09	.10000e+01	7	
5	.3100e-08	.2000e-08	.10000e+01	7	
6	.5100e-08	.2000e-08	.10000e+01	7	
7	.7100e-08	.2000e-08	.10000e+01	8	
8	.7350e-08	.2500e-09	.10000e+01	12	(*)
9	.8350e-07	.1000e-08	.10000e+01	36	
10	.1035e-07	.2000e-08	.10000e+01	19	
11	.1235e-07	.2000e-08	.10000e+01	9	
12	.1435e-07	.2000e-08	.10000e+01	7	
13	.1635e-07	.2000e-08	.10000e+01	12	
14	.1835e-07	.2000e-08	.10000e+01	13	
15	.2035e-07	.2000e-08	.10000e+01	10	
16	.2235e-07	.2000e-08	.10000e+01	11	
17	.2435e-07	.2000e-08	.10000e+01	11	
18	.2635e-07	.2000e-08	.10000e+01	14	
19	.2835e-07	.2000e-08	.10000e+01	11	
20	.3000e-07	.1650e-09	.10000e+01	12	

(*): The original time step can not converge within 50 iterations, with the reduced time step being used.

The coupling is the conductance of the emitter resistance re=1.

Table 5.6f

TTL inverters with re=1

(SLATE)

	time	time step	# of iterations
0	0	0	40
1	.5000e-10	.5000e-10	4
2	.1000e-09	.5000e-10	4
3	.3000e-09	.2000e-09	4
4	.1100e-08	.8000e-09	7
5	.3100e-08	.2000e-08	7
6	.5100e-08	.2000e-08	5
7	.7100e-08	.2000e-08	7
8	.9100e-08	.2000e-08	12
9	.1110e-07	.2000e-08	10
10	.1310e-07	.2000e-08	12
11	.1510e-07	.2000e-08	6
12	.1710e-07	.2000e-08	10
13	.1910e-07	.2000e-08	15
14	.2110e-07	.2000e-08	9
15	.2310e-07	.2000e-08	10
16	.2510e-07	.2000e-08	8
17	.2710e-07	.2000e-08	6
18	.2910e-07	.2000e-08	8
19	.3000e-07	.9000e-09	5

Table 5.7a

ECL inverters with rb=100

(SLATE-R)

	time	time step	coupling	# of iterations
0	0	0	.10000e-01	38
1	.7812e-12	.7812e-12	.10000e-01	4 (**)
2	.1562e-11	.7812e-12	.10000e-01	4
3	.1953e-11	.3906e-12	.10000e-01	3 (*)
4	.3615e-11	.1562e-11	.10000e-01	3
5	.4297e-11	.7812e-12	.10000e-01	3 (*)
6	.4687e-11	.3906e-12	.10000e-01	3 (*)
7	.6250e-11	.1562e-11	.10000e-01	3
8	.7031e-11	.7812e-12	.10000e-01	3 (*)
9	.7422e-11	.3906e-12	.10000e-01	3 (*)
10	.8984e-11	.1562e-11	.10000e-01	3
11	.9766e-11	.7812e-12	.10000e-01	3 (*)
12	.1016e-10	.3906e-12	.10000e-01	3 (*)
13	.1172e-10	.1562e-11	.10000e-01	3
14	.1250e-10	.7812e-12	.10000e-01	3 (*)
15	.1289e-10	.3906e-12	.10000e-01	3 (*)
16	.1309e-10	.1953e-12	.10000e-01	3 (*)
17	.1387e-10	.7812e-12	.10000e-01	3
18	.1426e-10	.3906e-12	.10000e-01	3

(**):

The original time step can not converge within 100 iterations, with the reduced time step being used.

(*): The original time step can not converge within 50 iterations, with the reduced time step being used.

The coupling is the conductance of the base resistance rb=100.

Table 5.7b

ECL inverters with rb=100

(SLATE)

	time	time step	# of iterations
0	0	0	12
1	.5000e-10	.5000e-10	3
2	.1000e-09	.5000e-10	3
3	.3000e-09	.2000e-09	4
4	.1100e-09	.8000e-09	4
5	.2000e-08	.9000e-09	4
6	.2050e-08	.5000e-10	4
7	.2100e-08	.5000e-10	3
8	.2300e-08	.2000e-09	3
9	.3100e-08	.8000e-09	5
10	.5100e-08	.2000e-08	5
11	.7100e-08	.2000e-08	4
12	.9100e-08	.2000e-08	4
13	.1110e-07	.2000e-08	3
14	.1310e-07	.2000e-08	3
15	.1510e-07	.2000e-08	3
16	.1710e-07	.2000e-08	3
17	.1910e-07	.2000e-08	3
18	.2110e-07	.2000e-08	3
19	.2310e-07	.2000e-08	3
20	.2510e-07	.2000e-08	3
21	.2710e-07	.2000e-08	3
22	.2910e-07	.2000e-08	3
23	.3000e-07	.9000e-09	3

5.6 Concluding Remarks

It is shown in Table 5.4 and Table 5.5 that the simulation results of the NMOS circuits and CMOS circuits are good and the time steps used in SLATE-R are identical with those in the SLATE program, because of the low coupling terms with the MOS technologies. CMOS circuits take slightly more iterations than NMOS circuits due to convergence, because of the higher signal gain and stronger coupling capacitance with the CMOS technology. From Table 5.6 and Table 5.7, it is found that for the bipolar transistor circuits, the number of iterations required for convergence is excessive because of the increased coupling and gain of the input circuits. By changing the emitter resistance in Table 5.6 to adjust the coupling, it clearly shows the effect of the coupling on the convergence speed. A limitation on the number of iterations per time point was set in the SLATE-R program. If the current time step can not reach convergence in a certain number of iterations, a new time step is chosen. Compared with the TTL circuits, the ECL circuits exhibit shorter gate propagation delay which causes the simulation of the ECL circuit to take more iterations and eventually result in very small time steps.

CHAPTER 6

THE SLATE-R PROGRAM

With the advantages mentioned in the previous chapters, the SLATE Program has been chosen to be the pedestal to implement the time-point relaxation techniques. The SLATE Program exploits the node tearing method which basically is just a generalized Norton equivalent circuit approach. Each subcircuit can be extracted as an equivalent circuit via the LU factorization. By clustering all the equivalent subcircuits to construct a simplified equivalent circuit, which is the interconnection matrix in Eq. (5.2), one can solve the interconnection matrix to obtain the node voltages of all the tearing nodes for each subcircuit. The next step is to go back to each subcircuit with the tearing node voltages as the stimuli to get the node voltage of the internal nodes. From the viewpoint of the matrix equations, the SLATE program is still inverting the whole matrix; all the subcircuits can be solved simultaneously. With this solution strategy, one need not worry about the order in which the subcircuits are to be processed, and the fan-in and fan-out nodes do not have to be identified.

However, the solution strategy of the relaxation techniques is to process each subcircuit individually by treating the fan-in nodes as the sources and the fan-out nodes as inputs to the next stage of subcircuits. Hence, one should identify the fan-in nodes and fan-out

nodes of each subcircuit and also determine the order in which the subcircuits are to be processed.

In this chapter, we will describe the implementation of the program SLATE-R (a Simulator with Latency and Tearing - Relaxed version). Different iterative schemes of the relaxation approaches are also described.

6.1 The Input Processing -- READIN Overlay

For the exploitation of the relaxation techniques, the fan-in nodes and the fan-out nodes of each subcircuit have to be identified. However, the circuit input format of the SLATE (SPICE2) program only gives the external nodes of each subcircuit without specifying the fan-in nodes and fan-out nodes. The first step of the program implementation is to change the SUBCKT card in the input file. The general form of the subcircuit definition in the SLATE program is as follows:

General form

.SUBCKT SUBNAM N1 <N2 N3 ...NE>

Examples:

.SUBCKT NAND2 10 20 30 40

In the subcircuit definition, SUBNAM is the subcircuit name, and N1, N2, ... ,NE are the external nodes. For example, the NAND2 circuit has four external nodes, node #10 is the DC power supply, node

#20 and node #30 are the input nodes while node #40 is the output node. In order to identify the fan-in nodes and the fan-out nodes, the SUBCKT card is modified to the form:

General form

.SUBCKT SUBNAM N1 <N2 N3 ... NE> NFIN

Examples:

.SUBCKT NAND2 10 20 30 40 3

where NFIN is the number of the fan-in nodes of the subcircuit. The DC power supplies are treated as the fan-in nodes in order to obtain the optimal reordering [10] performance. In this case, the NAND2 circuit has three fan-in nodes and one fan-out node, so NFIN = 3.

6.2 Analysis Sequencing — ERRCHK Overlay

In the ERRCHK overlay, the SLATE program constructs the node connection lists. This primitive and the event-driven algorithm in Section 5.3 are used to set up the order in which the subcircuits are to be processed.

6.3 Matrix Setup and Matrix Location — SETUP Overlay

With the exploitation of the node tearing approach, the SLATE program reorders all the tearing nodes to the border as shown in Eq. (5.1). The interconnection matrix shown in Eq. (5.2) is the core in

the solution strategies of the node tearing techniques. Because the relaxation techniques need not formulate the interconnection matrix, each subcircuit can be treated as one "independent" circuit with its fan-ins as the stimuli input voltage sources. Hence, the matrix set up is straightforward. The repetitiveness property of subcircuits is still used in the SLATE-R program, i.e., only one subcircuit description for each type of repetitive subcircuit need be stored and only one set of the submatrix sparse matrix pointers for each type of repetitive subcircuit is needed. In this overlay, a set of fan-in and fan-out lists are established to trace the input stimuli and the coupling terms among each subcircuit.

6.4 Analysis Procedure — DCTRAN Overlay in SLATE Program

6.4.1 Algorithms in SLATE Program

The analysis algorithms used in SLATE program are shown below:

```
        initialize;
        TIME = 0
        call SORUPD to set sources for the entire circuit;
        call ITER8;
        if (not converged) stop analysis;
           {   print operating-point solution;
           }
savout: store outputs
```

```
newtim: TIME = TIME + DELTA
        if (TIME > TSTOP ) exit;
           {   adjust DELTA for breakpoint table values;
               call SORUPD;
               call ITER8;
           }
        if (converged) goto tsterr;
           {   TIME = TIME - DELTA;
               DELTA = DELTA/8;
               goto tstdel;
           }
tsterr: call TRUNC;
        if (error acceptable) goto savout;
           {   TIME = TIME - DELTA;
               DELTA = DELNEW (computed in TRUNC);
           }
tstdel: if (DELTA < DELMIN ) stop analysis;
        goto newtim;
```

The actual Newton-Raphson iteration is controlled by subroutine ITER8. A flow chart for that subroutine follows.

6.4.2 Iteration Scheme in SLATE Program

```
        ITERNO = NONCON = 0 ;
```

```
        DONE = .false.;
        while (not done)
yload:  {   call YLOAD;
            if ((NOSOLV is nonzero) and
                (analysis = initial transient)) exit;
            load the circuit (except subcircuits);
            locx=locate(19); (load the first subcircuit)
subckl: if (locx = 0) goto sdcdcm ;
        latency check;
        (in time level or Newton-Raphson iteration level)
        if (nodplc(locx+9) is nonzero)   goto nxtckl;
           load elements in subcircuit;
nxtckl: locx = nodplc(locx);   (search for the next subcircuit)
        goto subckl;
sdcdcm: locx = locate(19);
subcks: if (locx = 0) exit;
        if (nodplc(locx+9) = 1) (latency in time level) goto nxtcks;
           call SDCDCM; (LU decomposition for subcircuit)
nxtcks: locx = nodplc(locx);   goto subcks;
        }
        ITERNO = ITERNO + 1
        if (ITERNO > iteration limit)  exit;
        if (NONCON = 0 )  DONE = .true.;
           {   call DCDCMP;
               (LU decomposition for the interconnection matrix)
           }
```

```
            {   call DCSOL; (solve the interconnection matrix)
                if (all the tearing nodes converged) NONCON = 0;
            }
        locx = locate(19)
solckt: if (locx = 0) goto yload;
        check latency flag;
        if ((nodplc(locx+9) is nonzero ) and
            (keep latency)) goto sdcsol;
            {   call SDCSOL;
                (solve the subcircuit with backward substitution)
                check convergence;
                if (not converged) NONCON = NONCON + 1
            }
sdcsol: locx = nodplc(locx);  goto solckt
        }
```

From the algorithms used in the SLATE program, it is found that with the implementation of the node tearing technique the SLATE program can take advantage of the latency exploitation. However, the node tearing approach used here is one special reordering technique which puts all the tearing nodes to the border of the system matrix. By solving the resulting interconnection matrix [8], one gets the external node voltages for each subcircuit, while the internal node voltages are solved by backward substitutions. From the viewpoint of solving the matrix equations of the system, full matrix inversion is

still needed in the SLATE program. In the next section, we describe how the relaxation techniques are implemented in the DCTRAN overlay and the core overlay of the SLATE program.

6.5 Modifications of the Analysis Procedure

--- DCTRAN Overlay in SLATE-R Program

Because each time only one subcircuit is analyzed with the use of the relaxation techniques, we just need to process the individual matrix equations for the corresponding subcircuit. This is a departure from the analysis procedure with the SLATE program. The analysis algorithms used in the SLATE-R program are shown below:

6.5.1 Algorithms in SLATE-R Program

```
        initialize;
        TIME = 0
        call ITER8;
        if (not converged) stop analysis;
           {   print operating-point solution;
           }
savout: store outputs
newtim: TIME = TIME + DELTA
        if (TIME > TSTOP ) exit;
           {   adjust DELTA for breakpoint table values;
               call ITER8;
```

```
            if (converged) goto tsterr;
            {   TIME = TIME - DELTA;
                DELTA = DELTA/8;
                goto tstdel;
            }
        }
tsterr: call TRUNC;
        if (error acceptable) goto savout;
        {   TIME = TIME - DELTA;
            DELTA = DELNEW (computed in TRUNC);
        }
tstdel: if (DELTA < DELMIN ) stop analysis;
        goto newtim;
```

Two different iteration schemes are implemented. In order to describe the iteration schemes, let us recall Eq. (3.6).

$$g_k(x_1^{n+1}, x_2^{n+1}, \ldots, x_{k-1}^{n+1}, x_k, x_{k+1}^{n}, \ldots, x_m^{n}) = 0 \qquad (6.1)$$

At time t_{n+1}, the k-th component of x^{n+1}, x_k^{n+1}, is obtained by solving the above scalar equation. In our implementation, one iteration scheme is that each time we only iterate one subcircuit once and go to the next subcircuit and continue this process until all the subcircuits are converged. The other scheme is to iterate each subcircuit until convergence is reached and then go to the next subcircuit. After all the subcircuits are processed, then go back to the

first subcircuit to check if the iteration has converged. This process proceeds until all the subcircuits are checked. The flow chart for the implementation of the first iteration scheme follows.

6.5.2 Iteration Scheme I in SLATE-R Program

```
        ITERNO = 0 ;
        DONE = .false.;
        while (not done)
        icheck = 0 (convergence flag);
iterat: locx = locate(19)
subckl: if (locx = 0)  goto ncheck;
           {   call SORUPD;
               (to set sources for this subcircuit)
           }
        {   call YLOAD;
            (to load elements for this subcircuit)
            if ((NOSOLV is nonzero) and
                (analysis = initial transient)) exit;
        latency check;
               (in time level or Newton-Raphson iteration level)
        if (nodplc(locx+9) is nonzero) exit;
           {   load elements in subcircuit;
               call SDCDCM;
               (LU decomposition for this subcircuit)
```

```
            }
         }
         if (nodplc(locx+9) is nonzero) goto nxtckl;
            {   call SDCSOL;
                (solve the subcircuit with backward substitution)
                NONCON = 0;
                check convergence; (for this subcircuit only)
                if (all the nodes of this subcircuit converged) NONCON = 0;
                   {   NONCON = NONCON + 1
                   }
                if (NONCON = 0)  goto nxtckl;
                   {   icheck = icheck + 1
                   }
            }
nxtckl: locx = nodplc(locx);
         (search for the next subcircuit)
         goto subckl;
ncheck: check convergence; (for the entire circuit)
         if (icheck = 0) DONE = .true.;
            {   ITERNO = ITERNO + 1
                if (ITERNO > iteration limit)  exit;
                icheck = 0;
                goto iterat
            }
```

The following is the flow chart for the implementation of the second iteration scheme:

6.5.3 Iteration Scheme II in SLATE-R Program

```
          IERNO = 0 ;
          DONE = .false.;
          while (not done)
          icheck = 0 (global convergence flag);
          ichckt = 0 (local convergence flag);
 iterat:  locx = locate(19)
 subckl:  if (locx = 0)  goto ncheck;
          ichckt = 0
          {   call SORUPD;
              (to set sources for this subcircuit)
          }
 yload:   {   call YLOAD; (to load elements for this subcircuit)
          if (ichckt is nonzero)  goto yload2;
          if ((NOSOLV is nonzero) and
              (analysis = initial transient)) exit;
          latency check;
          (in time level or Newton-Raphson iteration level)
          if (nodplc(locx+9) is nonzero) exit;
 yload2:  load elements in subcircuit;
          {   call SDCDCM (LU decomposition for this subcircuit);
          }
```

```
        }
        if (nodplc(locx+9) is nonzero) goto nxtckl;
        {   call SDCSOL;
            (solve the subcircuit with backward substitution)
        NONCON = 0;
        check convergence; (for this subcircuit only)
        if (all the nodes of this subcircuit converged) NONCON = 0;
           {   NONCON = NONCON + 1
               if (NONCON = 0)  goto nxtckl;
                  {   ichckt = ichckt + 1
                      icheck = icheck + 1
                      goto yload;
                  }
           }
nxtckl: locx = nodplc(locx); (search for the next subcircuit)
        goto subckl;
ncheck: check convergence; (for the entire circuit)
        if (icheck = 0) DONE = .true.;
           {   ITERNO = ITERNO + 1
               if (ITERNO > iteration limit)  exit;
               goto iterat
           }
```

The experimental results for different iteration schemes are given in the next section.

6.6 Experimental Results

Table 6.1 shows the results for the 11-stage chain of inverter circuits in Fig. 5.2.

Table 6.1

Simulation data for the 11-stage chain of inverter circuits

DC Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
5.63	3.20	3.30

Transient Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
39.72	52.92	16.82

Table 6.2 shows the results for the binary-to-octal decoder circuit in Fig. 5.3.

Table 6.2

Simulation data for the binary-to-octal decoder circuit

DC Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
11.87	7.85	6.27

Transient Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
49.88	61.65	14.10

Table 6.3 shows the results for the one-bit full adder circuit in Fig. 5.4.

Table 6.3

Simulation data for the one-bit full adder circuit

DC Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
6.18	5.25	4.67

Transient Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
31.95	48.56	17.47

Table 6.4 shows the results for the bipolar transistor cascade of inverters in Fig. 5.7.

Table 6.4

Simulation data for the bipolar transistor inverters

DC Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
2.62	3.52	2.12

Transient Analysis

CPU (sec)		
Scheme I	Scheme II	SLATE
14.45	33.07	8.42

6.7 Concluding Remarks

6.7.1 Remark I

From Table 6.1, Table 6.2 and Table 6.3, it is found that for the MOS digital circuits, Scheme II works faster than Scheme I does in the DC analysis, while in the transient analysis, Scheme II is slower than Scheme I. For example, for the 11-stage inverter chains, in the DC analysis Scheme II takes 3.20 seconds and Scheme I takes 7.23 seconds, whereas in the transient analysis Scheme II takes 52.92

seconds and Scheme I takes 39.72 seconds. On the other hand, from Table 6.4, for the bipolar digital circuit, in the DC analysis, Scheme II takes 3.52 seconds and Scheme I takes 2.62 seconds, while in the transient analysis, Scheme II takes 33.07 seconds and Scheme I takes 14.45 seconds. In other words, Scheme II is always slower than Scheme I in the simulation of the bipolar digital circuits.

Because there are no coupling capacitors for the MOS digital circuits in the DC analysis, in Scheme II one subcircuit is processed to convergence and then fed to the next subcircuit which obtains the exact stimuli to process the analysis, hence, there is no waste in the iterative process. For Scheme I the waveforms fed from the fan-in subcircuit into the next stage are not accurate until the fan-in subcircuit achieves convergence; before that, all the iterative processes do not make any sense.

However, in the transient analysis, from Table 5.5 it is found that the typical number of iterations needed to achieve convergence for MOS digital circuits is three or four at each time point. Let us define one unit as one iteration for one subcircuit. If we take a four cascade chain of uniform subcircuits as an example, then Scheme I takes four iteration sweeps to reach convergence with each sweep analyzing four subcircuits; then the total units needed for Scheme I are 16. If Scheme II is used, it takes four iterations for each subcircuit to reach local convergence, that means in the first run the total units are 16 and it takes at least two runs to ensure global convergence, that is, 32 units.

The strong coupling terms in the bipolar technology make the relaxation technique take more iterations to achieve convergence as shown in Table 5.6. The typical number of iterations needed to achieve convergence for bipolar transistor circuits is six to eight at each time point. Similarly, let us take the four cascade chain of uniform subcircuits as an example. For Scheme I it takes six sweeps to reach convergence, that is, 24 units. While Scheme II takes six iterations to achieve local convergence, each run needs 24 iterations. Scheme II needs at least two runs to ensure global convergence, that is, 48 units.

Our conclusion is that, if there is no coupling, then Scheme II which is used in MOTIS-C and PREMOS works very well just like MOTIS-C and PREMOS. When there is coupling, then Scheme I works faster than Scheme II does.

6.7.2 Remark II

In order to see the performance of the SLATE-R program in the simulation of strongly connected circuits, Table 6.5 and Table 6.6 show the results for the 3-stage ring oscillator in Fig. 4.1 and the SR Flip Flop in Fig. 6.1. Only Scheme I is used.

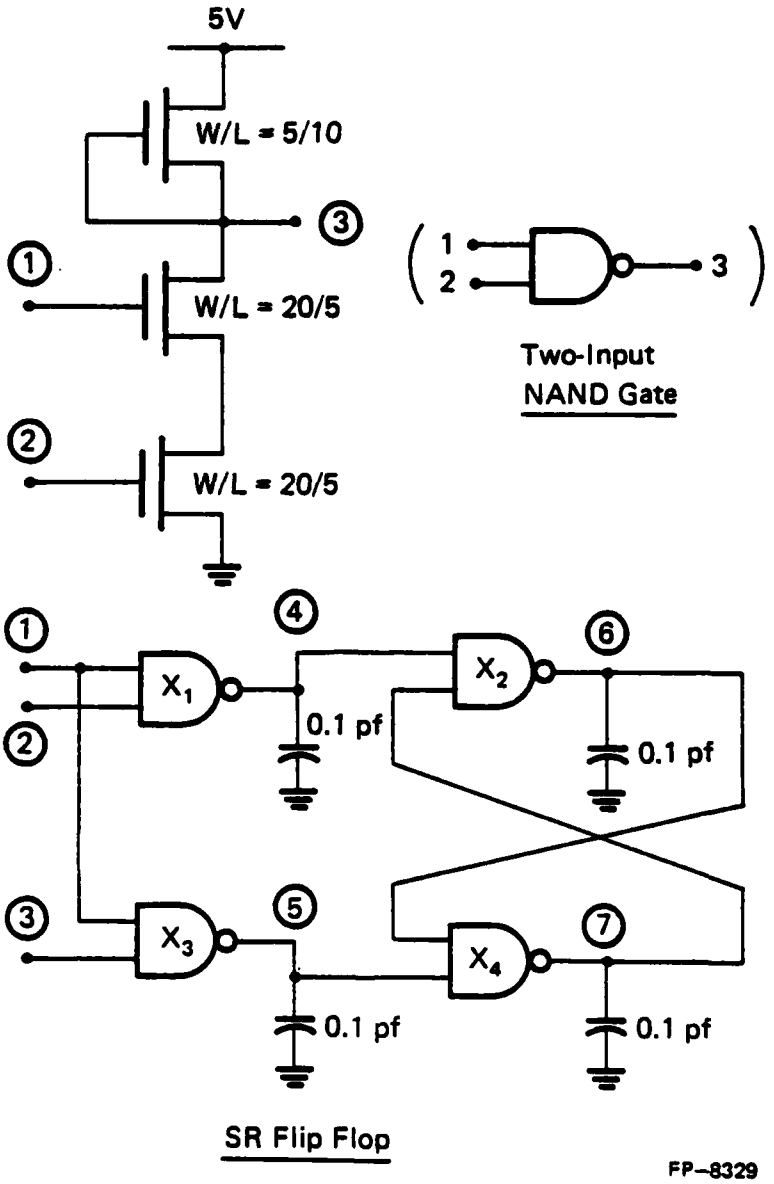


Fig. 6.1 SR Flip Flop.

Table 6.5 shows the results for the 3-stage ring oscillator in Fig. 4.1.

Table 6.5

Simulation data for the 3-stage ring oscillator

DC Analysis

CPU (sec)	
Scheme I	SLATE
2.13	0.92

Transient Analysis

CPU (sec)	
Scheme I	SLATE
18.82	9.48

Table 6.6 shows the results for the SR Flip Flop in Fig. 6.1.

Table 6.6

Simulation data for the SR Flip Flop.

DC Analysis

CPU (sec)	
Scheme I	SLATE
3.13	2.05

Transient Analysis

CPU (sec)	
Scheme I	SLATE
78.03	22.65

6.7.3 Remark III

It is found in Section 6.6 that the simulation speed of the SLATE-R program is two to three times slower than that of the SLATE program. In order to find which factors affect the simulation speed, the schemes of the latency exploitation in both SLATE-R and SLATE are removed to simulate some circuits again.

Table 6.7 shows the results for the SR Flip Flop in Fig. 6.1 without the latency exploitation.

Table 6.7

Simulation data for the SR Flip Flop circuit
(without the latency exploitation)

DC Analysis

	CPU (sec)	# of iterations
SLATE-R	3.50	25
SLATE	2.58	25

Transient Analysis

	CPU (sec)	# of iterations
SLATE-R	82.43	469
SLATE	26.58	171

Table 6.8 shows the results for the one-bit full adder circuit in Fig. 5.4 without the latency exploitation.

Table 6.8

Simulation data for the one-bit full adder circuit (without the latency exploitation)

DC Analysis

	CPU (sec)	# of iterations
SLATE-R	7.25	15
SLATE	5.03	15

Transient Analysis

	CPU (sec)	# of iterations
SLATE-R	41.45	84
SLATE	17.97	46

It is found that, for DC analysis, SLATE-R and SLATE take the same number of iterations. In Table 6.7, the CPU time for SLATE-R is 7.25 seconds and is 144.14% of the 5.03 seconds CPU time for SLATE. Thus, we can estimate the overhead for the relaxation method to predict the coupling is 44.14%. Similarly, in Table 6.8, the overhead to predict the coupling is 35.65%.

In transient analysis, Table 6.7 shows the SLATE-R takes 84 iterations in 41.45 seconds of CPU time, while SLATE takes only 46 iterations in 17.97 seconds. The CPU time spent in SLATE-R is 230.66%

of that spent in SLATE; the iteration number needed in SLATE-R is 182.61% of that needed in SLATE. The overhead to predict the coupling is 48.05%. In Table 6.8, the CPU time spent in SLATE-R is 310.12% of that spent in SLATE, and the iteration number needed in SLATE-R is 274.26% of that needed in SLATE. The overhead to predict the coupling is 35.86%.

From Table 6.7 and Table 6.8, we can determine the dominant factor which makes the difference in simulation speed. As we can see, more iterations are required for the relaxation method, and the overhead for the predictor is also a considerable factor.

CHAPTER 7

CONCLUSIONS

We used a simple linear circuit model to represent one gate driving another gate, and we investigated the numerical stability and convergence properties of the time-point (Gauss-Seidel) and waveform relaxation methods as a function of the amount of parasitic coupling capacitance between the gates and the gain of the driven gate in the active region.

In Chapter 3 our investigation shows that the modified Gauss-Seidel time-point relaxation method does not behave well for linear stiff systems if the stiffness is caused by the coupling. The method becomes numerically unstable when the size of the time step exceeds the smallest time constant in the system by a factor of three or more.

In Chapter 4 we apply the modified waveform relaxation method for the analysis of linear stiff systems. It is found that the iterations oscillate about the solution and an excessive number of iterations are required for convergence unless the time windows are approximately the size of the smallest time constant of the system. These results clearly indicate that if the system stiffness (strong Miller effect) is caused by the coupling, then the integration step size cannot exceed approximately the smallest time constant in the circuit in order to have good convergence properties.

However, practical implementation shows that both the waveform relaxation method and the modified Gauss-Seidel method are not affected very much by the pole-splitting phenomenon in the simulation of MOS digital circuits. This seems to be due to the low gain and weak coupling in these circuits. Also, digital MOS circuits are switched on and off fairly rapidly so that they are in the linear active region only a small percentage of the time interval.

In SLATE the time steps for transient analysis are controlled by the local truncation error only. However, because of the poorer stability convergence properties of the relaxation methods for some problems, the time step is controlled first by the local truncation error, but if the iteration limit is exceeded, the time step is reduced until good convergence is achieved. Thus, the SLATE-R program may not only require more iterations per time point, but also more time points may be required to obtain the solution in a given time interval.

It is shown in Table 5.4a and Table 5.4b, Table 5.5a and Table 5.5b that the time steps used in SLATE-R to analyze the MOS digital circuit examples in this thesis are identical with those in the original SLATE program. This means the time step is still controlled by the local truncation error in SLATE-R for these examples. However, for bipolar technology, our results show that it takes a tremendous number of iterations to achieve convergence. In some cases, the number of iterations exceed the iteration limit which forces the time step to be reduced and eventually causes the error message of "inter-

nal time step too small" to stop the execution. This means that when the coupling between subcircuits is too strong, the time steps are controlled by the iteration count instead of the local truncation error. Also, occasionally the wide oscillation between successive iterations causes the arithmetic operations to overflow before the iteration limit is reached, because of the strong coupling (terminal resistances) in the bipolar circuits. In Chapter 5, we used different terminal resistances for the bipolar TTL circuits and verified that the strong coupling causes the convergence problem.

Event driven techniques and latency exploitation are implemented in SLATE-R. Chapter 5 illustrates the effects of latency exploitation on savings of CPU time. The program structure is shown in Chapter 6. Two iteration schemes are discussed. It is found that the performance of different iteration schemes are coupling-oriented. For the cases of no coupling and strong coupling, Scheme II works better than Scheme I, while for weak coupling, such as the floating capacitor in MOS circuits, Scheme I is preferred. For very strong coupling, both Scheme I and Scheme II have poor numerical properties.

With the relaxation technique, there is no need to formulate the interconnection matrix like in the SLATE program. In the SLATE-R program, each subcircuit can be processed individually, which is very suitable for a multiprocessor computer system. Currently, we use the same time steps for all the subcircuits. Some advantage can be obtained if each subcircuit uses its own time step. However, the overhead for data management in a single processor system might

offset the algorithmatic advantage. The latency exploitation is used as a compensation for choosing the same time step for all the subcircuits. The major advantage of each subcircuit using its own time steps is to save CPU time with the penalty of data management; in this case, the latency exploitation scheme skips those latent subcircuits to save CPU time.

In this research, event-driven algorithms are implemented which sequence the subcircuits to be simulated on the basis of fan-in fan-out topology. Because the linked list data structure is being used in SLATE, the coupling terms are easily traced from the matrix pointer. With the exploitation of the modified Gauss-Seidel method, the coupling terms on each subcircuit are decoupled by using a predictor, such that the subcircuits can be solved one at a time.

It is found that SLATE-R is about a factor of two slower than SLATE. There are three factors that slow down the simulation speed: the first is the overhead of using predictor to decouple the coupling terms; the second is the larger number of iterations needed to get the convergence; and the third is that the tearing nodes and the power supply nodes are split for each subcircuit which induces a larger size of submatrix. In Section 5.2, the submatrix size for each subcircuit is 3x3; however, with the split of power supply nodes and the tearing nodes, the submatrix size is increased to 7x7, thus more CPU time is required for each subcircuit factorization. For future research, in order to speed up the simulation speed of SLATE-R, one can change the data structure such that all the voltage sources are

set in the first diagonal block exactly as Equation (5.4) shows, rather than splitting the voltage source nodes, and each subcircuit is solved with the submatrix which contains only the unknown variables. The other approach is to use individual time steps for each subcircuit. Currently SLATE-R and SLATE both use the same time steps for all the subcircuits and they are all solved with the smallest time step. In SLATE, all the subcircuits are processed through LU factorization as well as the interconnection matrix. Thus, in using a multiprocessor system to solve the equations, the LU factorization of the interconnection matrix could create a bottleneck. However, the solution strategy of the relaxation technique is to solve one subcircuit at a time with the coupling terms relaxed; therefore, it seems to have more potential for implementation in a multiprocessor computer system.

With the potential of taking full advantage of both the tearing technique and the relaxation technique, the implementation of the relaxation approach to multiprocessor systems will be a promising topic for future investigation. In order to implement the relaxation technique in the multiprocessor computer system, an automatic partition algorithm that separates the system into certain subsystems is very important. The basic idea will be to choose terminals with a minimum fan-out as the partitions. More specifically, choose those possible nodes, such as the gates in MOS circuits, as the set of candidates for partitioning, and try to partition this set into certain subsets with about the same size so as not to cause any bottleneck in the parallel process.

REFERENCES

[1] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory Report, no. ERL-M520, University of California, Berkeley, May 1975.

[2] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qassemzadeh and T. R. Scott, "Algorithms for ASTAP - a network analysis program," IEEE Transaction on Circuit Theory, vol. CT-20, pp. 628-634, November 1973.

[3] B. R. Chawla, H. K. Gummel and P. Kozak, "MOTIS - an MOS timing simulator," IEEE Transaction on Circuits and Systems, vol. CAS-22, pp. 901-910, December 1975.

[4] S. P. Fan, M. Y. Hsueh, A. R. Newton and D. O. Pederson, "MOTIS-C : a new circuit simulator for MOS LSI circuits," Proceedings 1977 IEEE International Symposium on Circuits and Systems, Phoenix, Arizona, pp. 700-703, April 1977.

[5] G. Arnout and H. De Man, "The use of threshold functions and boolean-controlled network elements for macromodeling of LSI circuits," IEEE Journal of Solid-State Circuits, vol. SC-13, pp. 326-332, June 1978.

[6] A. R. Newton, "The simulation of large scale integrated circuits," IEEE Transaction on Circuits and Systems, vol CAS-26, pp. 741-749, September 1979.

[7] N. B. G. Rabbat, A. L. Sangiovanni-Vincentelli and H. Y. Hsieh, "A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain," IEEE Transaction on Circuits and Systems, vol. CAS-26, pp. 733-741, September 1979.

[8] P. Yang, I. N. Hajj and T. N. Trick, "SLATE : a circuit simulation program with latency exploitation and node tearing," Proceedings IEEE International Conference on Circuits and Computers, pp. 353-355, October 1980.

[9] E. Lelarasmee, A. E. Ruehli and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale integrated circuits," IEEE Transaction on Computer-Aided Design, vol. CAD-1, no. 3, pp. 131-145, July 1982.

[10] Y. P. Wei, I. N. Hajj and T. N. Trick, "A prediction-relaxation-based simulator for MOS circuits," Proceedings IEEE International Conference on Circuits and Computers, pp. 34-37, September 1982.

[11] D.O. Pederson, "A historical review of circuit simulation," IEEE Transaction on Circuits and Systems, vol. CAS-31, no.1, pp103-111, January 1984.

[12] J. White and A. L. Sangiovanni-Vincentelli, "Relax2.1: a waveform relaxation based circuit simulation program," Proceedings IEEE Custom Integrated Circuits Conference, Rochester New York, pp. 232-236, May 1984.

[13] P.R. Gray and R.G. Meyer, Analysis and Design of Analog Integrated Circuits, John Wiley & Sons, New York, 1977.

[14] A. R. Newton and D. O. Pederson, "Analysis time, accuracy and memory requirement tradeoffs in SPICE2," IEEE Proceedings International Symposium on Circuits and Systems, pp. 6-9, 1978.

[15] G. D. Hachtel and A. L. Sangiovanni-Vancentelli,"A survey of third-generation simulation techniques," Proceedings of the IEEE, vol. 69, no. 10, October 1981.

[16] A.L. Sangiovanni-Vincentelli, "On the decomposition of large scale systems of linear algebraic equations," Proceedings of JACC, Denver, pp. 18-20, June 1979.

[17] J.A. George, " On block elimination for sparse linear systems," SIAM Journal of Numerical Analysis, vol.11, pp. 585-603, 1974.

[18] K. Sakallah and S.W. Director, "An activity-directed circuit simulation algorithm," IEEE Proceedings International Conference on Circuits and Computers, New York, pp. 1032-1035, October 1980.

[19] C. F. Chen, "The second generation MOTIS timing simulator -- an efficient and accurate approach for general MOS circuits," Proceedings IEEE International Symposium on Circuits and Systems, pp. 538-542, May 1984.

[20] R. Saleh and A. R. Newton, "Iterated timing analysis and SPLICE1," Proceedings IEEE ICCAD Conference, Santa Clara, Ca, September 1983.

[21] W.K. Chia, I.N. Hajj and T.N. Trick, "A survey of relaxation methods for the simulation of digital integrated circuits," Proceedings of the Midwest Symposium on Circuits and Systems, Puebla, Mexico, August 1983.

[22] W.K. Chia, T.N. Trick and I.N. Hajj, "Stability and convergence properties of relaxation methods for hierarchical simulation of VLSI circuits," Proceedings of the IEEE International Symposium on Circuit and Systems, Montreal, Canada, pp.848-851, May 1984.

[23] C.W. Gear, "Automatic multirate methods for ordinary differential equations," Information Processing 80, pp. 717-722, IFIP, 1980.

[24] T. Tarjan, "Depth-first search and linear graph algorithms," SIAM Journal of Computing vol. 1, no. 2, pp. 146-160, June 1972.

[25] A. E. Ruehli, A. L. Sangiovanni-Vincentelli and N. B. Rabbat, "Time analysis of large-scale circuits containing one-way macromodels," Proceedings IEEE International Symposium on Circuits and Systems, pp. 766-770, April 1980.

[26] N. Tanabe, H. Nakamura and K. Kawkita, "MOSTAP: a MOS circuit simulator for LSI circuits," Proceedings IEEE International Symposium on Circuits and Systems, pp.1035-1038, April 1980.

[27] P. Roth, Computer Logic, Testing and Verification, Computer Science Press, Potomac, MD, 1980.

[28] H. Y. Hsieh and N. B. Rabbat, "Computer-aided design of large networks by macromodular and latent techniques," Proceedings IEEE International Symposium on Circuits and Systems, pp. 688-692, April 1977.

VITA

Wei-Kong Chia was born in Tainan, Taiwan, China on October 16, 1954. He received his B.S. and M.S. degrees in 1976 and 1978, respectively, both in electrical engineering from National Cheng Kung University in Tainan, Taiwan. During the academic year 1978-1979, he was a lecturer with the Electronic Engineering Department at Fu-Jen University in Taipei, Taiwan. He pursued one year graduate study at Lehigh University at Bethlehem, PA, in 1979-1980. In June 1980 he entered the University of Illinois. From June 1980 to August 1984 he was a teaching assistant and research assistant with the Electrical and Computer Engineering Department and the Coordinated Science Laboratory. His research interests are in the areas of computer-aided design, circuits and systems, semiconductor device modeling and solid-state electronics.

END

FILMED

1-86

DTIC